# Motion Planning for an Autonomous Terrestrial Vehicle in Static Environments

Andries Jacobus Bester

Electronic Systems Laboratory, Department Electrical and Electronic Engineering,
Stellenbosch University, Stellenbosch, South Africa, 7600


UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY


Supervisor:

Dr. C.E. van Daalen

*Thesis presented in partial fulfilment of the requirements for the degree of Master in Engineering at Stellenbosch University*

March 2018

UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

## Plagiaatverklaring / Plagiarism *Declaration*

1 Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
*Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.*

2 Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
*I agree that plagiarism is a punishable offence because it constitutes theft.*

3 Ek verstaan ook dat direkte vertalings plagiaat is.
*I also understand that direct translations are plagiarism.*

4 Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
*Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.*

5 Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
*I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.*

March 2018

# Abstract

Ground or land-based vehicles capable of navigating autonomously in various environments are rapidly becoming more popular and have multiple advantages and uses, from a Mars rover to autonomous agricultural vehicles or search-and-rescue robots. Vehicle manufacturers are also racing to be the first to deliver a completely self-driving car, with technological advances already facilitating features like self-parking, predictive braking, adaptive cruise control and traffic sign recognition. This project aims to contribute to autonomous navigation research by improving on motion planning techniques for terrestrial vehicles.

The work done includes the modelling and control of a quad bike with actuators installed to facilitate autonomous control of the steering, throttle and brakes of the vehicle. With the simulation model of the vehicle, the commanded behaviour of the vehicle could be quantised in sets of movements to produce replicable manoeuvres. The path planning algorithm developed in this project then searches in a map of the environment and uses these manoeuvres to build a path for the vehicle, which adheres to the kinematic and dynamic constraints of the vehicle. To supplement the modular design of the path planning algorithm, a simplified conflict detection algorithm and map representation were created to allow the path planner to be developed without knowledge of the map or the representation of obstacles.

The results obtained from the field tests indicate that the path planning algorithm could find a path that became more optimal the longer the algorithm was allowed to search, and that the vehicle could follow the path generated by the planner. The functionality of the path planner was demonstrated without the ability to do replanning in real time, and is therefore advised that further research should include the real-time application and optimisation of the proposed algorithm.

# Opsomming

Grond- of landgebonde voertuie wat in staat is om outonoom te navigeer in verskeie omgewings, is besig om vinnig meer populêr te raak en het vele voordele en gebruike, van 'n Mars-karretjie tot outonome landbouvoertuie of soek-en-reddingstuie. Voertuigvervaardigers is ook besig om mekaar te jaag om die eerste motor op te lewer wat volledig self kan bestuur, waar tegnologiese verwikkelinge reeds funksies fasiliteer soos self parkeer, voorspelbare remtoepassing, aanpasbare spoedbeheer en die herkenning van verkeerstekens. Hierdie projek se doel is om 'n bydrae te lewer tot die navorsing van outonome navigasie deur die verbeter van tegnieke om die beweging van landgebonde voertuie te beplan.

Die werk wat gedoen is, sluit die modellering en beheer van 'n vierwielmotorfiets in waarvan die remme, stuurwiel en brandstofsneller automaties beheer kan word deur voorheen bygevoegde aktueerders. Met die simulasiemodel van die vierwielmotorfiets kon die beheerde gedrag saamgevat word in 'n stel van bewegings wat herhaalbaar is. Die padbeplanningsalgoritme, wat ontwikkel is in hierdie projek, soek 'n roete op die kaart van die omgewing wat gebou kan word, soos gedefinieer deur die bewegings van die vierwielmotorfiets. Sodoende kan dit 'n pad produseer wat voldoen aan die dinamiese bewegingsbeperkinge van die motorfiets. Om die modulêre ontwerp van die padbeplanner te ondersteun, moes 'n vereenvoudigde metode om 'n botsing of konflik vas te stel en 'n kaartvoorstelling ontwikkel word sodat die padbeplanner afgesonder kan word van hoe 'n kaart of hindernis voorgestel word.

Die toetsresultate wat verkry is uit die praktiese toetse, het aangedui dat die padbeplanner wel 'n pad kon vind in die kaart van die omgewing en dat die pad al hoe meer optimaal raak, hoe langer die algoritme toegelaat word om aan te hou soek. Die toetsresultate het ook bewys dat die vierwielmotorfiets in staat was om die beplande roete te volg en voltooi. Die funksionaliteit van die padbeplanner is gedemonstreer sonder dat die beplanner die vermoë het om in werklike tyd 'n nuwe roete aan te pas. Dit word dus voorgestel dat toekomstige navorsing die toepassing van herbeplanning in werklike tyd moet insluit en die ontwikkelde algoritme verder te optimeer.

# Acknowledgements

I would like to thank my project supervisor, Corné van Daalen, for his open door policy and continuous support, eagerness to help, thorough explanation of concepts and his inspiration during the final stages of the write-up.

Many thanks to my family, friends and fellow students for their motivation and understanding for the time spent on this project.

# Contents

# List of Figures

# List of Tables

# Nomenclature

### Abbreviations

| | |
|---|---|
| ft | Feet |
| Hz | Hertz |
| km | Kilometres |
| km/h | Kilometres per hour |
| m | Metres |
| m/s | Metres per second |
| ms | Milliseconds |
| N | Newtons |
| Nm | Newton meter |
| rad | Radians |
| rad/s | Radians per second |
| rev/min | Revolutions per minute |
| s | Seconds |

### Acronyms

| | |
|---|---|
| 2D | Two-dimensional |
| 3D | Three-dimensional |
| A* | A-star search algorithm |
| ACB | Actuator Control Board |
| ATV | All Terrain Vehicle |
| CVT | Continuously variable transmission |
| D* | D-star search algorithm |

| | |
|---|---|
| DGPS | Differential Global Positioning System |
| EKF | Extended Kalman Filter |
| EM | Expectation-Maximization |
| GPS | Global Positioning System |
| GUI | Graphical user interface |
| hOBC | Higher on-board controller |
| LPM | Local Planning Method |
| OBC | On-board controller |
| PD | Proportional and Differential control |
| PI | Proportional and Integral control |
| PRM | Probabilistic Roadmap Method |
| RRT | Rapidly-exploring Random Tree |
| RRT* | Rapidly-exploring Random Graph with Tree characteristics |
| SLAM | Simultaneous Mapping and Localization |

**Symbol Conventions**

| | |
|---|---|
| $x$ | Scalar |
| $\mathbf{x}$ | Vector |
| $\mathbf{x}(t)$ | Time-varying vector |
| $X$ | Set |
| $\mathbf{X}$ | Matrix |
| $\mathbf{X}(t)$ | Time-varying matrix |
| $\mathbf{X}(t, \omega)$ | Vector of random processes |
| $\overline{x}$ | Mean of $x$ |

**List of Symbols**

| | |
|---|---|
| $\alpha_{front}$ | Percentage lateral slip of the front wheels |
| $\alpha_{rear}$ | Percentage lateral slip of the rear wheels |

| | |
|---|---|
| $\beta$ | Angle of the velocity vector relative to the heading |
| $\eta_{gears}$ | Friction loss coefficient of the gears |
| $\eta_{slip}$ | The clutch slip loss coefficient |
| $\lambda$ | Percentage longitudinal slip between the driving wheels and the terrain |
| $\lambda_{max}$ | Percentage slip at which the peak longitudinal slip coefficient occur |
| $\mu_{lat,slip}(\ )$ | Friction coefficient as a function of the percentage lateral slip |
| $\mu_{long,slip\_max}$ | Peak longitudinal slip coefficient for a type of terrain |
| $\mu_{long,slip}(\ )$ | Friction coefficient as a function of the percentage longitudinal slip |
| $\mu_{roll}$ | Friction coefficient of the terrain |
| $\dot{\omega}_w$ | Rotational acceleration of the rear wheels |
| $\omega_C$ | Rotational speed at the output of the large sprocket |
| $\omega_e$ | Rotational speed of the engine in rad/s |
| $\omega_G$ | Rotational speed at the output of the electric motor gearbox |
| $\omega_m$ | Rotational speed of the steering motor output shaft |
| $\omega_w$ | Rotational speed of the rear wheels |
| $\omega_{vehicle}$ | Rotational speed of the vehicle about its z-axis |
| $\dot{\psi}$ | Rate of heading change of the vehicle |
| $\psi$ | Heading angle of the vehicle relative to earth axis |
| $\Psi$,$\Theta$,$\Phi$ | Yaw , pitch and roll relative to the inertial axis system |
| $\Psi_{track}$ | The track heading |
| $\tau_{throttle}$ | The percentage throttle opening |
| $\rho_a$ | Air density |
| $\dot{\theta}$ | The measured angular speed of the steering wheels |
| $\dot{\theta}_{feed-forward}$ | The feed-forward signal |
| $\theta$ | Steering angle of the front wheels |
| $\theta_1$ | The starting angle of a radial length |

| | |
|---|---|
| $\theta_2$ | The stopping angle of a radial length |
| $\theta_{ref}(s)$ | Reference steering angle of the steering control system |
| $a$ | Horizontal stretch of the ellipse |
| $A_f$ | Frontal area of the vehicle |
| $A_{ca}$ | Contact area |
| $b$ | Vertical stretch of the ellipse |
| $B_e$ | The viscous friction coefficient |
| $c_d$ | Aerodynamic drag coefficient |
| $c_{conv}$ | Conversion factor from rev/min to rad/s |
| $D(s)$ | The controller in a time domain control system representation |
| $D_{crosstrack}(s)$ | The cross-track error controller in a time domain control system representation |
| $E_{dest}$ | The destination East coordinate |
| $E_{src}$ | The source East coordinate |
| $F(s)$ | The filter in a time domain control system representation |
| $F_{braking}$ | The braking force acting on the disc brake |
| $F_{cent}$ | The centrifugal force in the clutch |
| $F_{cornering}$ | Cornering force responsible for centripetal acceleration |
| $F_{drag}$ | Aerodynamic drag force |
| $F_{lat,front}$ | Lateral force acting on the front wheels |
| $F_{lat,rear}$ | Lateral force acting on the rear wheels |
| $F_{rolling}$ | Opposing force due to rolling resistance of the terrain |
| $F_{traction}$ | Maximum forward driving force achievable and is a function of the percentage slip. |
| $FF(s)$ | The feed-forward filter in a time domain control system representation |
| $g$ | Acceleration due to gravitational force |
| $G(s)$ | The plant in a time domain control system representation |

| | |
|---|---|
| $F_1$ | Focal point 1 of the ellipse |
| $F_2$ | Focal point 2 of the ellipse |
| $\bar{I}$ | Average current draw for a specified time |
| $I_a$ | Current drawn by the electric actuator |
| $I_{safe}$ | The safe operating current for a specified time |
| $J_e$ | the equivalent engine inertia |
| $J_w$ | Equivalent wheel and axle inertia |
| $J_{vehicle}$ | Inertia of rotation of the vehicle about its z-axis |
| $K$ | The gain in a time domain control system representation |
| $L_a$ | Inductance of the electric actuator |
| $l_{front}$ | Distance of the front axle from the centre of gravity of the vehicle |
| $l_{rear}$ | Distance of the rear axle from the centre of gravity of the vehicle |
| $L_{track}$ | Ground track length |
| $m_{front}$ | Mass on the front wheels |
| $m_{rear}$ | Mass on the rear wheels |
| $m_{vehicle}$ | Total mass of the vehicle |
| $N_e$ | The speed of the engine in rev/min |
| $N_{b(max)}$ | The engine speed at which peak engine power |
| $N_{dest}$ | The destination North coordinate |
| $N_{src}$ | The source North coordinate |
| N, E, D | North, East and Down axes of the inertial axis system |
| $p$ | The local contact pressure |
| $p_1$ | Point 1 |
| $p_2$ | Point 2 |
| $p_a$ | Pressure |
| $P_b$ | Power produced by the engine |

| | |
|---|---|
| $P_N$,$P_E$,$P_D$ | Positions in the North, East and down axes directions |
| $P_{b(max)}$ | The peak engine power |
| $\bar{r}$ | The effective radius at which the braking force is applied on the disc brake |
| $R(s)$ | Reference signal in a time domain control system representation |
| $R_a$ | Resistance of the electric actuator |
| $r_i$ | The inner radius of a radial width |
| $R_K$ | Combined CVT and gear ratio |
| $r_o$ | The outer radius of a radial width |
| $r_w$ | Radius of the rear wheels |
| $r_{curve}$ | Instantaneous radius of the path of the vehicle during turning |
| $R_{CVT}$ | Ratio of the CVT |
| $R_{gears\_F}$ | The forward driving ratio of the gears |
| $R_{gears\_R}$ | The reverse driving ratio of the gears |
| $R_{gears}$ | Ratio of the gears |
| $S$ | Set of states $s$ |
| $s_{goal}$ | End point of the planning problem |
| $s_{start}$ | Start point of the planning problem |
| $T_C$ | Torque at the output of the large sprocket |
| $T_e$ | Torque produced by the engine |
| $T_G$ | Torque at the output of the electric motor gearbox |
| $T_m$ | Torque produced by the steering motor |
| $T_p$ | The engine torque corresponding to the maximum engine power |
| $T_w$ | Torque produced by the engine applied on the wheels |
| $T_\psi$ | Turning moment of the vehicle |
| $T_{align}$ | Self-aligning torque on the front wheels |
| $T_{braking}$ | Torque applied on the wheels resulting from braking |

| | |
|---|---|
| $T_{ca}$ | Torque at the contact area |
| $T_{clutch\_in}$ | The input torque to the clutch |
| $T_{clutch\_out}$ | The output torque of the clutch |
| $T_{e(max)}$ | The maximum engine torque |
| $T_{load}$ | The load torque at a given point |
| $T_{WL}$ | The resulting actuation torque on the front left wheel |
| $T_{WR}$ | The resulting actuation torque on the front right wheel |
| $\dot{v}_{long}$ | Longitudinal acceleration of the vehicle |
| $V_a$ | Voltage applied to the electric actuator |
| $V_N$,$V_E$,$V_D$ | Velocities in the North, East and down axes directions |
| $v_{lat}$ | Lateral velocity of the vehicle |
| $v_{long}$ | Longitudinal velocity of the vehicle |
| $x_{track}$ | The projected distance of the vehicle along the track length from the source waypoint |
| x,y,z | x,y,z axes of the body axis system |
| $Y(s)$ | Output signal in a time domain control system representation |
| $y_{error}$ | The vehicle's cross track error |

# Chapter 1

# Introduction

Ground or land-based vehicles capable of navigating autonomously in various environments are rapidly becoming more popular and have multiple advantages and uses. From search-and-rescue robots and reconnaissance vehicles in the defence industry, to automated warehouse carts, to cars and trucks that drive in and between cities or towns, the possible applications of autonomous navigation is vast.

Vehicle manufacturers are racing to be the first to deliver a self-driving car. Some technologies that are currently in cars that provide some autonomous functionality include: self-parking, adaptive cruise-control, collision detection and predictive braking, traffic sign recognition and lane assist using torque steering.

This project does not aim to develop the next self-driving car, but rather to contribute to autonomous navigation research and application. At Stellenbosch University, in the Department of Electrical and Electronic Engineering, an autonomous navigation research group (AutoNav) was established for the purpose of such research, and facilitated the project.

## 1.1 Project definition

Using the project topic and breaking it down to it's key elements, a better understanding of what the project implies might be acquired.

**Motion Planning**

The first part of the topic implies some form of **path finding and planning** will be implemented. This is the ability to find a route from one point to another, or to plan where to go next. To be able to find a route, an **obstacle detection and avoidance** component must be in place to detect if something is in the way or predict if a collision will occur, and then planning to avoid this to ensure safe driving from one point to another. The success of the planned motion of a vehicle will depend on how effectively that vehicle can be controlled. Therefore, a reliable **control system** forms a crucial part of the topic description to ensure that the goal point can be reached.

**Autonomous terrestrial vehicle**

This part of the topic gives an indication of the type of vehicle on which the developments and research results will be implemented, namely, an unmanned ground vehicle. To be more specific, a four wheeled motorbike, or commonly known as a **quad bike**, is used to demonstrate the autonomous ground vehicle research projects on. This quad bike has been modified to have full autonomous capabilities: actuators and an on-board computer controlling the vehicle inputs like throttle, brakes, steering and gear selection have been installed.

**Static environments**

This part of the topic implies that all obstacles are stationary and no replanning will be required to avoid unexpected or moving obstacles along the route. It is also assumed that an exact map of the environment is available from the start to do path planning on.

Using partially mapped environments (where an exact map of the environment is not available) is a more practical application of the project, but due to time constraints of the project and unforeseen work on the control system of the vehicle, it was decided to only use completely mapped environments to test the path finding algorithm. Partially mapped environments imply that some of the areas in the map might be unexplored or contain no information of what is behind objects that obscured the line of sight of the vehicle's mapping sensors.

## 1.2 Autonomous Navigation

Autonomous navigation of a vehicle, be it aerial, terrestrial or aquatic, implies that after being given a destination to reach, no human input is required for the vehicle to reach its goal. The project forms part of the AutoNav research on autonomous navigational systems. Such a system is complex to implement, and can be broken down into smaller modules that integrate to achieve autonomous navigation. This modular design of the autonomous navigation framework allows for the development of each separate module independently from one another, as long as the interfacing requirements for each module is clearly defined. It is therefore necessary to understand the framework of the autonomous navigation system used for all AutoNav projects and how this project fits into it.

The autonomous navigation framework developed by the AutoNav group is shown in Figure 1.1. It starts in the top right corner with user inputs. The most basic input would be location information of a goal point to be reached. Additional information can be accommodated, for example a specific time that the goal point should be reached or a specific orientation that the vehicle should be in at the goal point.

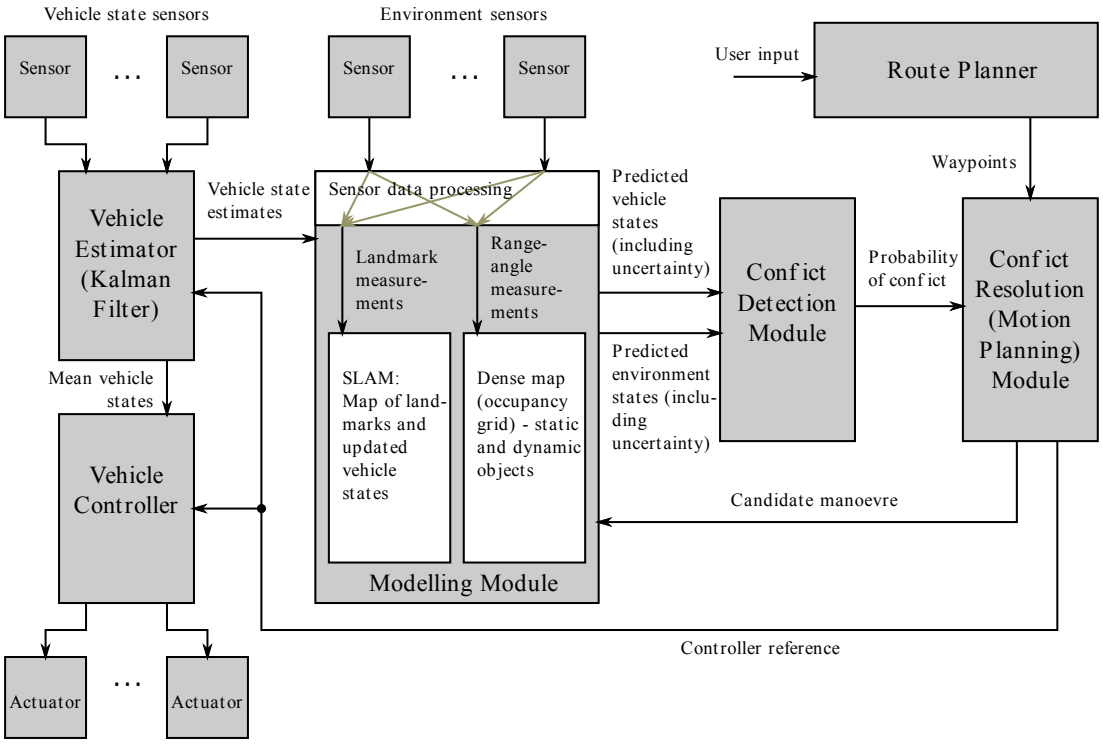



Figure 1.1: Autonomous navigation system framework

The user inputs are fed into the Route Planner which processes it to generate waypoints or milestones to be reached. The Route Planner can be seen as a pre-planner which is executed before the vehicle starts to move. The next module in line is the Motion Planning Module, which is tasked with solving the problem of how the vehicle should move from one point to the next. In applications where constant replanning is required, this module is also referred to as the Conflict Resolution Module.

The Motion Planning Module supplies the Modelling Module with a candidate manoeuvre, or set of movements, that would let the vehicle reach a point towards the next waypoint. The Modelling Module contains the map of the environment, and constantly updates it with information from the vehicle's mapping or environment sensors. The candidate manoeuvre is projected onto the map and passed along to the conflict detection module which checks for a collision or conflict. A collision means that the distance between the vehicle and the obstacle is zero, i.e. they collide. Conflict is when the distance between the vehicle and obstacle is less than a certain threshold, and includes a collision. If no conflict is detected, the candidate manoeuvre is accepted and the Motion Planning Module sends the set of reference control instructions to the Vehicle Controller to execute the manoeuvre using the vehicle actuators.

For the autonomous vehicle to know where it is in relation to the reference path, a Vehicle Estimator is required. The Vehicle Estimator also receives the reference control inputs and measures the actual movement of the vehicle by combining all sensor data to produce the

best estimate of the current vehicle states. A vehicle state is a combination of position, velocity and orientation information to represent the vehicle for a specific time interval. These estimated states are presented to the Modelling Module to chart the movement of the vehicle in the map.

The cycle is repeated between the Motion Planning, Modelling and Conflict Modules to produce a path between waypoints, all the way to the end goal. For applications that require a dynamic map, this is done in real-time while the vehicle is moving. For applications that implement a static map, this process can be executed before the vehicle starts to move.

## 1.3 Project Objectives

By combining the requirements as set out in Sections 1.1 and 1.2, and following a bottom-up approach, we identify the following objectives to reach the project aim:

- Modelling the dynamics, kinematics and behaviour of the vehicle to incorporate it in the navigation system, to be able to better plan a path that the vehicle can execute.
- Improving the effectiveness and reliability of the control systems of the vehicle, in order to better control it to follow a calculated path.
- Set up static environmental maps of actual terrain to execute a planned path and to test the navigation system practically.
- Developing a conflict detection algorithm that can analyse obstacles defined in the static environmental maps and test candidate paths for conflict or collision.
- Developing a motion planning algorithm.
- Combining and implementing the above mentioned into the AutoNav framework to test the motion planning of the vehicle.

## 1.4 Thesis Overview

With a basic understanding of the aim of the project, an overview of the following Chapters are provided to indicate the flow of work to be presented in the thesis.

Chapter 2 presents the test vehicle on which the research will be implemented, discussing each hardware and software component. It also distinguishes between previous work done and defines the scope of work for this project.

Modelling and control system design is done in Chapter 3. The chapter starts by providing some background information on vehicle modelling and derivation techniques. It then continues to describe the vehicle's longitudinal and lateral dynamics. The design of the vehicle speed and orientation controllers follows thereafter.

In Chapter 4 the concept of manoeuvres is introduced and the significance thereof. Previous manoeuvres used on the vehicle are mentioned and further developments are described. The local path planner, the algorithm that uses the manoeuvres to find a path segment from one point towards the next, is also covered.

Chapter 5 presents the theory on mapping methods and ways of representing the environment to the autonomous navigation system. The background of conflict detection is discussed in general, however, focus is placed on the deterministic method, which is the chosen method of conflict detection for this project.

Path planning is discussed in Chapter 6 and a detailed literature study is done on motion planning algorithms including; grid based search methods, potential field, geometric methods and random sampling.

In Chapter 7 the implementation of the autonomous navigation system and testing procedures is discussed. The chosen and adapted planning method from Chapter 6 is described and the limitations and advantages are defined.

Chapter 8 provides field results and analysis of the implemented autonomous navigation system on three different types of terrain: tar road, gravel road and a grass field. Some simulations are also discussed to illustrate the algorithm's performance on various map scenarios.

The thesis is concluded in Chapter 9 with an overview of the controllers and path planner performance, algorithm and implementation efficiency, the possibility of using partially mapped environments and implementing replanning on-the-go. Some applications of the research on self-driving cars are also presented.

# Chapter 2

# Test Vehicle

The test vehicle used for the project is presented in this chapter. It starts by describing the test vehicle, followed by how it was automated and then provides a representation of the control software architecture. Lastly, the chapter concludes with the scope of the project.

## 2.1 Vehicle Overview

The test vehicle used for the project is a yellow and black Kymco MXU150 quad bike. It was purchased by the Department of Electric & Electronic Engineering of Stellenbosch University for the purpose of implementing and testing autonomous navigation research on. The most significant vehicle properties are summarised from the Kymco Service Manual [9] in Table 2.1.

Table 2.1: Vehicle properties

| **KYMCO MXU150** | |  | |
|---|---|---|---|
| Engine capacity | 149.9 cc | | |
| Idle Speed | 1700 rpm | Brakes - Front | Dual drum |
| Transmission | CVT with reverse | Brakes - Rear | Single disc |
| Clutch type | Centrifugal | Braking Distance | $\leq$ 20.6 m |
| CVT gear ratio | 2.8-0.95 | Minimum turning radius | 3 m |
| Chain drive ratio | 7.226 | Maximum steer angle | $\pm$ 40° |
| Reverse gear ratio | 26.902 | Overall length | 1775 mm |
| Weight - Front wheel | 87 kg | Overall width | 950 mm |
| Weight - Rear wheel | 88 kg | Overall height | 1040 mm |
| Total weight | 175 kg | Wheel base | 1115 mm |

Normally, a human driver would be sitting on the seat of the quad bike and execute control over the various control inputs to the bike. For instance, if it is required to accelerate the

bike, the driver would actuate the throttle lever. Similarly, if deceleration is required, the driver would actuate the front brake lever and step on the rear brake pedal. To change the direction in which the bike is driving, the driver can select between the 'Forward' and 'Reverse' gears. Lastly, the heading in which the bike is driving can be changed by the driver who turns the steering wheel.

To automate the quad bike, the human driver is replaced by actuators that can execute control to the above mentioned control inputs of the quad bike. Section 2.2 details the additional hardware installed to achieve automation of the quad bike.

## 2.2 Component Overview

The predecessor to this project, *Automation and Navigation of a Terrestrial Vehicle*, by W. Visser [56], included the automation of the quad bike. This section outlines the additional hardware installed by Visser to automate it. In Figure 2.1 the actuators and their position on the quad bike are indicated.

The throttle actuator is a servo motor that is installed in line with the throttle cable between the lever on the handlebar and the carburettor on the engine. The radial movement of the servo shaft is converted to linear movement in the cable with a short lever arm, where the linear travel is short enough to approximate it by the length of the arc at the tip of the lever.

The gear selection and braking actuators are electric motors with gearbox units, and are implemented on the same principle as the throttle actuator with a lever attached to the shaft of the motor unit.

Lastly, the steering actuator is also an electric motor with gearbox unit. The output shaft of the motor unit drives a small sprocket that in turn drives a chain link that is connected to a larger sprocket attached to the shaft between the steering wheel and the steering links. The motor gearbox uses a worm gear, which allows for rotational power to be transferred from the input shaft to the output shaft, but not from the output shaft to the input shaft. This means that any rotational force on the output shaft will be locked in place and not transferred back to the motor. Therefore, the steering wheel cannot be turned while the motor is stationary. The larger sprocket on the steering shaft is held in place by a removable shear pin. This pin is a safety and convenience feature. The shear pin can be broken in an emergency if the human driver has to change the heading of the quad bike while being autonomously controlled. The pin would also be removed when the quad bike is transported between storage and test locations, to allow a human driver to steer the bike.

As shown in Figure 2.1, the remaining hardware includes an additional battery, wheel encoder, emergency stop, controller electronics and a Lidar.

The additional battery was installed to supply electric power to the controller electronics and all the actuators. The emergency stop switch is used to switch off the engine in an emergency. When activated, it cuts the electric supply to the spark plug. The Lidar is a sweeping laser sensor that is used by the Mapping Module to enable the quad bike to "see". The wheel encoder is used to measure the rotational speed proportional to the wheel speed of the quad bike. The indicated controller electronics consists of the on-board controller, GPS module, accelerometer, gyroscope and magnetometer. The unlabelled grey box located above the gear actuator, in Figure 2.1, contains the actuator control board that executes low level control of the actuators.


Figure 2.1: Test Vehicle (source: W.Visser [56])

## 2.3 Software Overview

The software architecture implemented on the automated quad bike is distributed among the actuator control board (ACB), on-board controller (OBC), an on-board laptop and a

ground station computer, illustrated by Figure 2.2.

The ground station computer is used in all of the AutoNav autonomous vehicle research applications. It deploys the Differential Global Positioning System (DGPS), receives telemetry data from the test vehicle and can remotely enable and disable the autonomous navigation system.

The vehicle laptop, also called the higher on-board computer (hOBC), is ruggedised and mounted on the quad bike. It is used to facilitate the drive-by-wire capability via a joystick and to execute the autonomous navigation algorithms when the autonomous navigation controller is enabled. In the AutoNav framework shown in Figure 1.1, the Route Planner, Motion Planning, Conflict Detection and Modelling modules are run on the vehicle laptop. It receives the vehicle state estimates and other vehicle sensor data from the OBC, and sends the control data to the OBC.

The OBC was developed by the AutoNav group and is used on all of the vehicles in the autonomous navigation research. In the AutoNav framework shown in Figure 1.1, the Vehicle Estimator is implemented on the OBC in the form of a kalman filter, which is discussed in greater detail in Section 3.2. It combines multiple sensor inputs for the estimator, listed in Table 3.1, and receives the control data from the hOBC. The OBC also executes the outer control loops of the Vehicle Controller, detailed in Sections 3.4 and 3.5, by providing control references to the ACB. Lastly, the OBC controls the throttle actuator directly.

Due to the size of actuators on the test vehicle, an additional three ACB's were designed by Visser [56] to manage the inner control loops of the Vehicle Controller. Each ACB controls one of the three electric motor units used as the steering, braking and gear shift actuators. The ACB receives the reference control inputs from the OBC and provides feedback to the OBC.

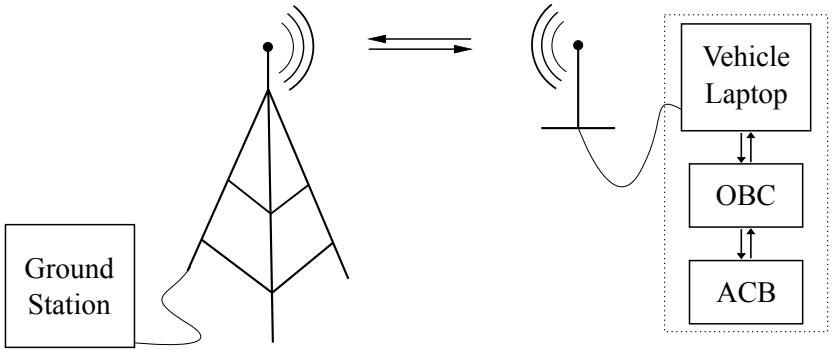



Figure 2.2: Vehicle software architecture

## 2.4 Scope of Work

With an understanding of the test vehicle and the implemented hardware components and software frameworks, the scope of this project can be presented to distinguish between previous work done, concurrent projects and this project.

A lot of time and effort was spent on improving the vehicle simulation model, evident from the length of Chapter 3, even though it was not the main focus of this project. The intention was to contribute to the in-house developments of the AutoNav group and future projects to follow.

Starting from the bottom of the software architecture indicated in Figure 2.2, on the ACB, the steering motor controller was redone to improve the control of the steering motor position and rate of change. One of the three actuator control boards was previously damaged, and due to the unavailability of the circuit board designs, it was not replaced. It was decided to omit the control of the gear shift lever and to use the remaining two actuator control boards to control the steering mechanism and the brakes of the vehicle.

On the OBC, the throttle actuator control was redone to improve the vehicle speed controller. The Vehicle Controller state machine by Visser [56] was also not used and a new state machine was developed specifically for this project. The outer control loop of the steering controlled was revised and adapted to use manoeuvre control more effectively. The OBC control logic was modified to include a data buffer to receive the control inputs calculated by the vehicle laptop of the path to be executed. The Vehicle Estimator was used as is and did not require additional work. The logging script of the OBC was modified to include additional controller states and measurements for the purpose of controller design and path following evaluation.

The graphical user interface (GUI) program on the vehicle laptop was adapted from a concurrent project and used to facilitate the communications between the laptop and the OBC. The GUI is written in C++ and in this project was modified to create an interface between the path planner implemented in MATLAB (on the vehicle laptop) and the OBC. Future work would be to convert the MATLAB code of the path planner to C++ and include it in the GUI program. For the duration of the project, this GUI was continuously revised and re-purposed for each test, starting from the controller design phase until the path planning and execution phase.

As mentioned, the path planner was implemented in MATLAB and contains the Motion Planning, Conflict Detection and Mapping Modules of the AutoNav framework. The Motion Planning Module is unique to this project and forms a large part of the work done. The Conflict Detection Module was simplified for the purpose of this project as discussed in Subsection 5.2.1. The existing Mapping Module was completely removed due to the Lidar sensor that was not functioning at the time of the project. The Mapping Module

was replaced by preloaded and static, fully mapped environments and did not include the ability to sense the environment in real-time. Any safety risks during autonomous testing was resolved by the human rider present on the vehicle.

# Chapter 3

# Vehicle Modelling and Control

As mentioned in previous sections, the project is based on a **four wheeled ground vehicle**. The quad bike is sometimes referred to in literature works as an ATV (all terrain vehicle), which is a category of vehicles capable of driving on- and off-road and which is not limited to vehicles with only four wheels. Due to the diversity of all terrain vehicles, it was difficult to find literature based specifically on quad bike modelling.

Detailed vehicle performance simulation is not the aim of this part of the project, but rather to develop and implement a control system which is robust enough to compensate for non-linearities and reject unpredicted vehicle behaviour differences outside a predetermined bound. Simple car dynamics are assumed at first, and more complex dynamics are then added as the need arises for more accurate simulation to be applicable for design of the controllers.

The methods of deriving the equations describing a mechanical system are not limited to using the sum of forces and moments technique. For interest, it was found in literature that the use of a Bond graph to derive the equations of a physical dynamic system is not considered unusual. Bond graphs were invented by a professor at MIT in 1959, Henry Paynter, and is based on the flow of energy in a system, the same as Kirchhoff's current and voltage laws. The bond graph method was first published in 1961, by Paynter, in a book called *Analysis and Design of Engineering Systems* [46]. The paper by Hung and Hong 2004 [25] uses bond graphs to derive the dynamics of a key mechanical system of the quad bike transmission system, which is described later on in this Chapter. In this project only the sum of forces and moments technique is used to model the mechanical systems.

In this chapter the axis systems are introduced to provide a mathematical reference to the dynamics of the vehicle. It is then followed by the Vehicle Estimator with descriptions of the sensors used to estimate the vehicle's position and orientation relative to the environment. The chapter continues to discuss the longitudinal dynamics, modelling and control of the vehicle, followed by the lateral dynamics, modelling and control.

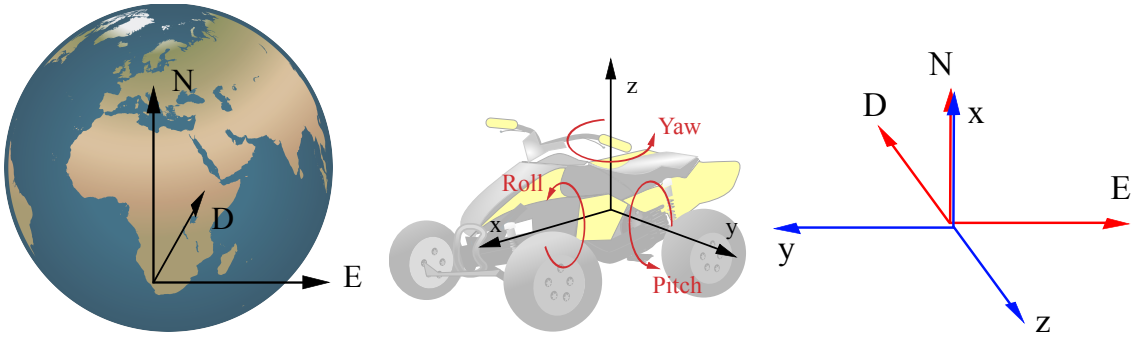
## 3.1 Axis System

To mathematically describe the vehicle dynamics, an axis system should be defined. The body axis is fixed to the body of the vehicle, and moves with the vehicle. The movement of the vehicle (i.e. the vehicle states, and how they change) is usually described in the inertial axis system.

The inertial axis system, referred to as the $NED$ axes (North-East-Down) in Figure 3.1(a), describes the global axes with respect to the earth's surface and follows the right handed orthogonal axis system. $N$ points due North, $E$ points East and $D$ points down (perpendicular to the local horizontal) and perpendicular to the north and east axis. The $NED$ axis system assumes a flat, non-rotating earth, and the origin can be chosen to coincide with some convenient reference point on the earth's surface. The influence of the earth's rotation is not of importance to the land based vehicle, because it is restricted to the earth's surface.

The body axis system, referred to as the $xyz$ axes in Figure 3.1(b), describes the axes of the vehicle body, and also follows the right handed orthogonal axis system. The body axis origin coincides with the vehicle's centre of mass, which is estimated to be in the centre of the vehicle. This also allows for the simplified representation of the vehicle as a point to the path planner and conflict detection modules, as explained in Section 5.2.1. The $x$-axis points in the forward direction of the vehicle, $y$ points to the left and $z$ points vertically upwards. Position information from the GPS unit and vehicle dynamics relative to the centre of gravity will be translated to the body axis system.

If the vehicle in Figure 3.1(b) was standing flat with its reference point at the origin of the inertial axis system in Figure 3.1(a) and facing directly North, the comparison of the two axis systems can be shown by Figure 3.1(c).




(a) Inertial axis (b) Body axis (c) Comparison of the inertial and body axes

Figure 3.1: The different axis systems used

Furthermore, the rotation of the body axis relatively to the inertial axis can be described by small rotations of Euler 3-2-1 angles. Rotation about the $z$-axis is given by the angle

$\Psi$ to describe the yaw of the vehicle. Rotation about the $y$-axis is given by the angle $\Theta$ to describe the pitch of the vehicle. Lastly, the rotation about the $x$-axis is given by the angle $\Phi$ to describe the roll of the vehicle.

## 3.2 Vehicle State Estimator

For the vehicle to know its position and orientation relative to the environment, it utilises a combination of sensors to continuously estimate the vehicle states. This estimation function is implemented on the on-board controller in the form of a real-time extended Kalman filter (EKF). The EKF combines the sensor data to achieve a least-squares best estimate of the current vehicle state at a set time interval. In this project the EKF is updated every 20 ms with new measurement data from the sensors listed in Table 3.1. The estimated vehicle state vector is

$$x = \begin{bmatrix} P_N & P_E & P_D & V_N & V_E & V_D & \Psi & \Theta & \Phi \end{bmatrix}^T,$$

consisting of three positions, three velocities and three angles in the inertial axis system.

Table 3.1: Estimator sensors and description

| **Sensor** | **Description** |
|---|---|
| Accelerometer | 3-axis acceleration measurement in the body axis, corrected for the gravitational vector |
| Gyroscope | 3-axis angular velocity measurements about the body axis |
| Magnetometer | 3-axis approximate orientation relative to magnetic North |
| DGPS | 3-axis position and velocity measurements in the inertial axis |
| Wheel encoder | approximate forward speed in the body axis, subjected to wheel slip |

## 3.3 Vehicle Dynamics

For this project, a reliable and concise source describing the kinetics and dynamics of a vehicle similar to a quad bike in state space representation, was not found during the literature study. All car-like models are too complex or overly simplified, or only longitudinal or only lateral dynamics are considered. The following equations were derived from basic principles using the book by G. Genta, *Motor Vehicle Dynamics: Modeling and Simulation* [19], and are used in the simulation model of the test vehicle.

### 3.3.1 Longitudinal Dynamics

The longitudinal dynamics of a vehicle describe the forward or reverse trajectory that a vehicle would follow while driving. In Figure 3.2, the rear half of a vehicle is illustrated with the driving forces and moments applied on one of the rear wheels. The reader is referred to Table 3.2 for a summary of variables and their respective descriptions used in the equations to follow.

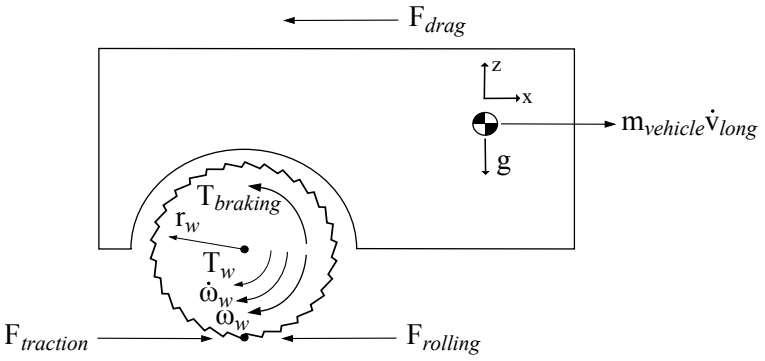



Figure 3.2: Longitudinal forces and moments diagram of the rear half of a vehicle

Using the sum of forces in the longitudinal direction the equation describing the longitudinal acceleration of the vehicle can be given as

$$m_{vehicle}\dot{v}_{long} = F_{traction} - F_{drag} - F_{rolling}$$

where $F_{traction}$ is the maximum forward driving force at the point of contact between the wheels and the driving surface, influenced by the amount of slipping that occurs between the wheels and the driving surface. The wind resistance on the vehicle is represented by $F_{drag}$ and $F_{rolling}$ represents the rolling resistance of the vehicle on the driving surface.

Using the sum of moments about the centre of the rear wheel, the equation describing the rotational acceleration of the rear wheels is

$$J_w\dot{\omega}_w = T_w - T_{braking} - r_w(F_{traction} - F_{rolling})$$

where $T_w$ is the driving torque applied on the wheels by the engine and drive train components. The braking torque applied on the wheels is given by $T_{braking}$ and $r_w$ defines the approximates outer radius of the wheels.

For the above two acceleration equations the following were used:

$$F_{traction} = \mu_{long,slip}(\lambda)m_{rear}g$$

where $\mu_{long,slip}$ is the friction coefficient as a function of longitudinal slippage. The downward force on the rear wheels is represented by the mass $m_{rear}$ and gravitational vector $g$. The function describing the longitudinal slip coefficient is presented in the book by Genta [19] as

$$\mu_{long,slip}(\lambda) = \frac{2\mu_{long,slip\_max}\lambda_{max}\lambda}{\lambda_{max}^2 + \lambda^2}$$

with the slip defined to be

$$\lambda = \begin{cases} \frac{\omega_w r_w - v_{long}}{|v_{long}|} & \text{for } v_{long} \neq 0 \\ 0 & \text{for } v_{long} = 0 \end{cases}.$$

The aerodynamic drag equation is used to calculate the wind resistance of the vehicle,

$$F_{drag} = \frac{1}{2}\rho_a c_d A_f v_{long}^2$$

with the front area $A_f$ of the vehicle approximated by a flat rectangular surface for this project. The rolling resistance of the vehicle is

$$F_{rolling} = \mu_{roll} m_{vehicle} g$$

where $\mu_{roll}$ is the friction coefficient of the terrain, assumed to be constant and independent of the speed of the vehicle.

These equations will be used in the simulation model of the quad bike to design the vehicle speed controllers, as well as in the path planner algorithm to ensure the calculated path adheres to the dynamic constraints of the vehicle.

Table 3.2: Longitudinal dynamics variables

| **Variable** | **Description** |
|---|---|
| $F_{traction}$ | Maximum forward driving force achievable and is a function of the percentage slip. |
| $F_{drag}$ | Aerodynamic drag force |
| $F_{rolling}$ | Opposing force due to rolling resistance of the terrain |
| $T_w$ | Torque produced by the engine applied on the wheels |
| $T_{braking}$ | Torque applied on the wheels resulting from braking |
| $\mu_{roll}$ | Friction coefficient of the terrain |
| $\mu_{long,slip}(\ )$ | Friction coefficient as a function of the percentage longitudinal slip |
| $\lambda$ | Percentage longitudinal slip between the driving wheels and the terrain |
| $\mu_{long,slip\_max}$ | Peak longitudinal slip coefficient for a type of terrain |
| $\lambda_{max}$ | Percentage slip at which the peak longitudinal slip coefficient occur |
| $m_{vehicle}$ | Total mass of the vehicle |
| $m_{rear}$ | Mass on the rear wheels |
| $v_{long}$ | Longitudinal velocity of the vehicle |
| $\dot{v}_{long}$ | Longitudinal acceleration of the vehicle |
| $\omega_w$ | Rotational speed of the rear wheels |
| $\dot{\omega}_w$ | Rotational acceleration of the rear wheels |
| $J_w$ | Equivalent wheel and axle inertia |
| $r_w$ | Radius of the rear wheels |
| $\rho_a$ | Air density |
| $c_d$ | Aerodynamic drag coefficient |
| $A_f$ | Frontal area of the vehicle |
| $g$ | Acceleration due to gravitational force |

### 3.3.2 Lateral Dynamics

The lateral dynamics of a vehicle describes the sideways and rotational movement of a vehicle while turning. In Figure 3.3, a simplified top-down view of the vehicle is illustrated with the forces and turning moments applied. The reader is again referred to Table 3.3 for a summary of variables and their respective descriptions used in the equations to follow.

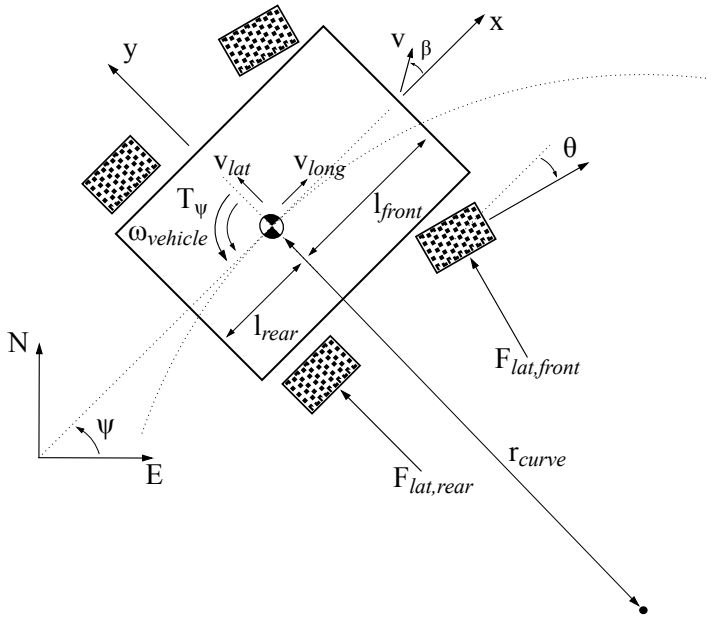



Figure 3.3: Lateral forces diagram

The equations describing the direction of movement $\beta$ and the radius $r_{curve}$ of the circle when making a turn can be given by

$$\beta = \arctan(\frac{v_{lat}}{v_{long}})$$

and

$$r_{curve} = F_{cornering} m_{vehicle} v_{long}^2,$$

where $v_{lat}$ is the sideways velocity in $y$-axis direction and $v_{long}$ the longitudinal velocity in the $x$-axis direction of the body axis system. The force responsible for centripetal acceleration is $F_{cornering}$ and $m_{vehicle}$ is the total mass of the vehicle.

The equation describing the rate of change of the heading of the vehicle can be given by

$$\dot{\psi} = \frac{T_\psi}{J_{vehicle}}$$

where $T_\psi$ is the turning moment and $J_{vehicle}$ the vehicle's rotational inertia around the $z$-axis of the body axis system.

For the above equations the following were used:

$$T_\psi = F_{lat,front} l_{front} \cos\theta - F_{lat,rear} l_{rear}$$

The sum of moments is given in one direction by the lateral force $F_{lat,front}$ on the front wheels at a length $l_{front}$ from the vehicle's centre of mass corrected by angle $\theta$ and the opposite direction by the lateral force $F_{lat,rear}$ on the rear wheels at a length $l_{rear}$ from the vehicle's centre of mass.

$$F_{lat,rear} = \mu_{lat,slip}(\alpha_{rear}) m_{rear} g$$
$$F_{lat,front} = \mu_{lat,slip}(\alpha_{front}) m_{front} g$$

The lateral forces on the front and rear wheels acting at the point of contact between the wheel and the terrain is dependent on the lateral slip friction coefficient $\mu_{lat,slip}$, the mass on the rear wheels $m_{rear}$ and front wheels $m_{front}$ respectively, and the gravitational vector $g$. The cornering force can be determined by the sum of lateral forces on the front and rear wheels:

$$F_{cornering} = F_{lat,front} \cos\theta + F_{lat,rear}$$

In the book by Genta [19], the percentage lateral slip at the front and rear wheels are distinguished by

$$\alpha_{rear} = \arctan \frac{v_{lat} + \omega_{vehicle} l_{rear}}{|v_{long}|}$$
$$\alpha_{front} = \arctan \frac{v_{lat} - \omega_{vehicle} l_{front}}{|v_{long}|} - \theta \text{sign}(v_{long})$$

where $\omega_{vehicle}$ is the rotational speed of the vehicle about the $z$ body axis.

These equations will be used in the simulation model of the quad bike to design the steering controllers, as well as in the path planner algorithm to ensure the calculated path adheres to the dynamic constraints of the vehicle.

Table 3.3: Lateral Dynamics variables

| **Variable** | **Description** |
|---|---|
| $F_{lat,rear}$ | Lateral force acting on the rear wheels |
| $F_{lat,front}$ | Lateral force acting on the front wheels |
| $F_{cornering}$ | Cornering force responsible for centripetal acceleration |
| $T_{\psi}$ | Turning moment of the vehicle |
| $\mu_{lat,slip}(\ )$ | Friction coefficient as a function of the percentage lateral slip |
| $\alpha_{rear}$ | Percentage lateral slip of the rear wheels |
| $\alpha_{front}$ | Percentage lateral slip of the front wheels |
| $\beta$ | Angle of the velocity vector relative to the heading |
| $\theta$ | Steering angle of the front wheels |
| $r_{curve}$ | Instantaneous radius of the path of the vehicle during turning |
| $v_{lat}$ | Lateral velocity of the vehicle |
| $\psi$ | Heading angle of the vehicle relative to earth axis |
| $\dot{\psi}$ | Rate of heading change of the vehicle |
| $\omega_{vehicle}$ | Rotational speed of the vehicle about its z-axis |
| $J_{vehicle}$ | Inertia of rotation of the vehicle about its z-axis |
| $l_{rear}$ | Distance of the rear axle from the centre of gravity of the vehicle |
| $l_{front}$ | Distance of the front axle from the centre of gravity of the vehicle |
| $m_{front}$ | Mass on the front wheels |

## 3.4 Vehicle Speed Control

In order to control the longitudinal velocity of the vehicle, an understanding of the propulsion and drive train dynamics should be acquired to model the vehicle.

The following vehicle elements in the drive train need to be considered in the simulation model of the quad bike: engine, transmission, clutch, final drive, brakes and wheels. Each element is discussed, and the equations of motion describing it. With elements where the

mathematical model is too complex, an approximation is used in the simulation model for this project, and references to sources are provided which attempt to better describe these elements. The schematic shown in Figure 3.4 illustrates the drive train elements of the vehicle.

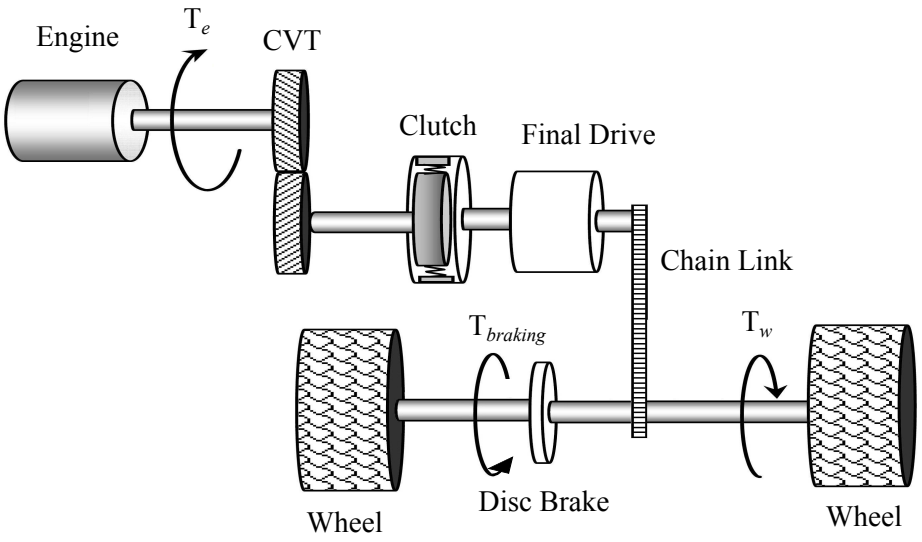



Figure 3.4: Schematic of the quad bike drive train

Power delivered to the wheels start at the engine which is connected to the transmission. In the case of the quad bike, this is a continuously variable transmission (CVT). The CVT is connected to a clutch, which is then followed by the final drive. The final drive facilitates the selection between forward and reverse driving, and has different fixed gear ratio's for forward and reverse respectively. From the final drive a chain link connects to the rear axle of the vehicle. On the rear axle the rear brakes and wheels are mounted.

### 3.4.1 Drive train elements

In this subsection a description and modelling of each drive train element shown in Figure 3.4 follows.

#### Internal Combustion Engine

In an internal combustion engine, torque is produced by the combustion of fuel and air which drives a piston and crank mechanism. The ratio of air and fuel determines the combustion force, and by implication the resulting torque produced. The amount of fuel in the mixture is controlled by the throttle opening. The relation between throttle opening, torque produced and engine speed is dependent on numerous factors like engine dimensions, type of fuel used, oxygen quality of the air, method of fuel injection, internal friction and

atmospheric pressure, just to name a few. Due to this complexity, engine dynamics are almost never derived from basic principles. The engine can be characterised by doing a dynamometer test on the actual vehicle, and then curve fitting the results to acquire an empirical equation. The Department of Mechanical and Mechatronic Engineering of Stellenbosch University has a dynamometer, but after some enquiries it was found that the device is too large for the quad bike's short wheel base. Again, because the purpose of the project is not to exactly model the quad bike engine, no further effort was used on finding a suitable dynamometer test platform.

In the literature study done on engine modelling, two equations were found which attempt to describe the engine torque produced by any engine, given the maximum torque or power and corresponding engine speed is known. The first equation is

$$T_e = \frac{1000 P_b}{c_{conv} N_e}$$

where

$$P_b = \left( A \frac{N_e}{N_{b(max)}} + B \left( \frac{N_e}{N_{b(max)}} \right)^2 - C \left( \frac{N_e}{N_{b(max)}} \right)^3 \right) P_{b(max)} \tau_{throttle}.$$

The engine torque [N.m] is given by $T_e$, $P_b$ is the engine power [kW] and $\tau_{throttle}$ is the percentage throttle opening. The engine speed is given by $N_e$ [rev/min] and $N_{b(max)}$ is the engine speed at which peak engine power $P_{b(max)}$ is achieved. Constants $A$, $B$ and $C$ are used for weighting and influence the form of the curve. The factor $c_{conv}$ is used to convert from engine speed in rev/min to rad/s. This equation is an empirical formula suggested by the book, *Theory of Machines and Mechanisms*, by Uicker et al. [55].

The second equation is

$$T_e = \left( T_{e(max)} - \left( \frac{T_{e(max)} - T_p}{(N_{b(max)} - N_{e(max)})^2} \right) (N_e - N_{e(max)})^2 \right) \tau_{throttle},$$

where $T_e$ is the engine torque and $Throttle$ is the percentage throttle opening. $T_{e(max)}$ is the maximum engine torque and $N_{e(max)}$ is the engine speed at which the maximum torque is achieved. $T_p$ is the engine torque corresponding to the maximum engine power and $N_{b(max)}$ is the engine speed at which the maximum engine power occurs. This equation is used in the publication by Dragos et al. [13].

Figure 3.5 shows the resulting curves for the two equations when used with the quad bike's engine parameters of 10 N.m at 5000 rev/min, and 8 kW at 7500 rev/min. As it can be seen, the equation used by Dragos et al. is more conservative with an almost constant torque over the range of engine speed.

By looking at Figure 3.4, the second-order differential equation that governs the engine speed can be derived, assuming no losses in any of the components between the wheels

and engine:

$$J_e\dot{\omega}_e + B_e\omega_e = T_e - \frac{T_w + T_{braking}}{R_{CVT}R_{gears}}$$

where $\omega_e$ is the engine speed in rad/s, $J_e$ is the equivalent engine inertia and $B_e$ is the viscous friction coefficient. $T_e$ is the torque produced by the engine, $T_w$ is the load torque as imposed by the road on the wheels and $T_{braking}$ is the braking torque applied on the wheels. $R_{CVT}$ and $R_{gears}$ are the gear ratios of the continuously variable transmission and the final drive respectively.

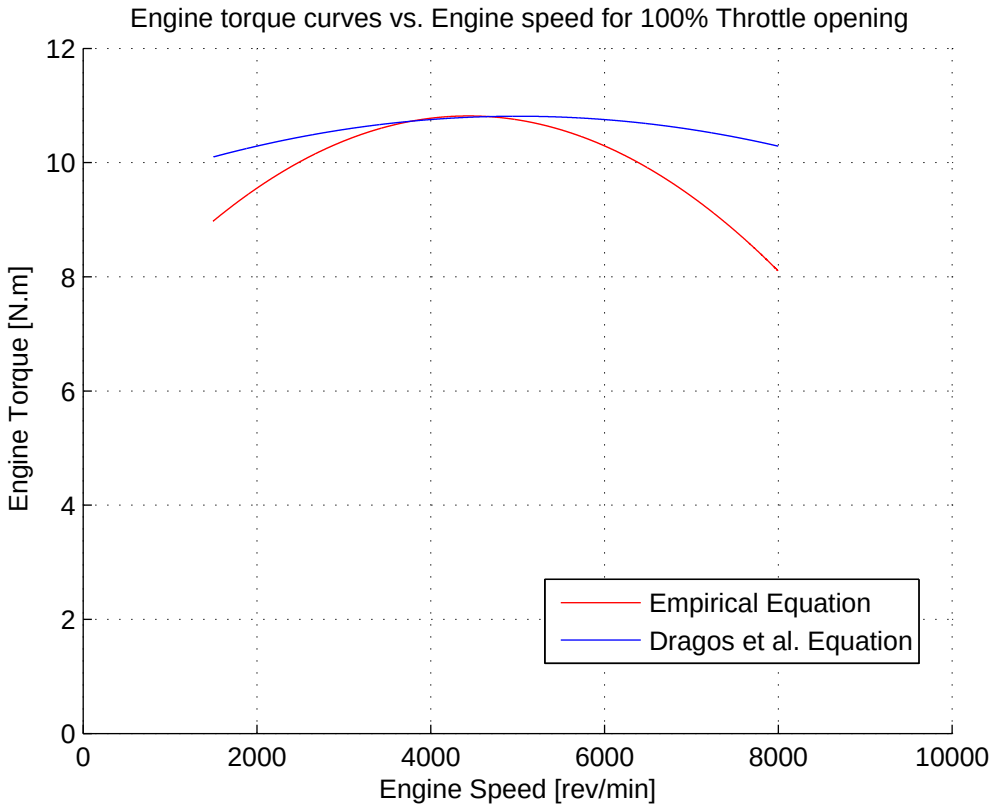



Figure 3.5: Comparing two different engine model equations

### Continuously Variable Transmission

The next drive train element following the engine is the CVT, from Figure 3.4, which is connected to the driveshaft of the engine. The CVT is a transmission that can change continuously through an infinite number of effective gear ratios between maximum and minimum values. This contrasts with other mechanical transmissions which have discrete gear ratios. Different types of variable transmissions are used in the automotive industry today, for example, variable-diameter pulley, toroidal traction drive, hydrostatic and friction cone CVTs. According to the seminar by Kevin Lang in 2000, [34], the history of a continuously variable transmission dates back to the 15th century, when Leonardo Da

Vinci conceptualised the first CVT in 1450. Due to technology constraints, the first CVT was only implemented in a vehicle in the 1950s by a Netherlands firm, DAF. CVTs are found today in vehicles manufactured by most major automotive leaders including Subaru, Nissan, VW, Honda, Audi, Ford, Toyota, BMW, Dodge, Mitsubishi and Suzuki.

The quad bike has a variable diameter pulley type CVT, which consists of a speed sensing driving pulley, a torque sensing driven pulley, and a rubber V-belt joining the two pulleys. Figure 3.6, adapted from the publication by G. Julio and J. Plante [29], presents a sectioned view of the CVT. The pulleys are split perpendicular to their axes of rotation. The workings of the CVT is described by [29] as follows:

The gear ratio is changed by moving the two halves of one pulley closer together and the two halves of the other pulley farther apart. Due to the inner walls of the pulleys being slightly conical and the V-shaped cross section of the belt, this causes the belt to ride higher on one pulley and lower on the other. Doing this changes the effective diameters of the pulleys, which in turn changes the overall gear ratio. The distance between the pulleys does not change, and neither does the length of the belt, so changing the gear ratio means both pulleys must be adjusted (one bigger, the other smaller) simultaneously in order to maintain the proper amount of tension on the belt. Both the driving and driven pulleys are subjected to axial and torque loading during operation. This implies that the rotating speed of the driving pulley together with the road load acting on the driven pulley determines the gear ratio uniquely. In the quad bike the CVT ratio is limited between 0.95 and 2.8.

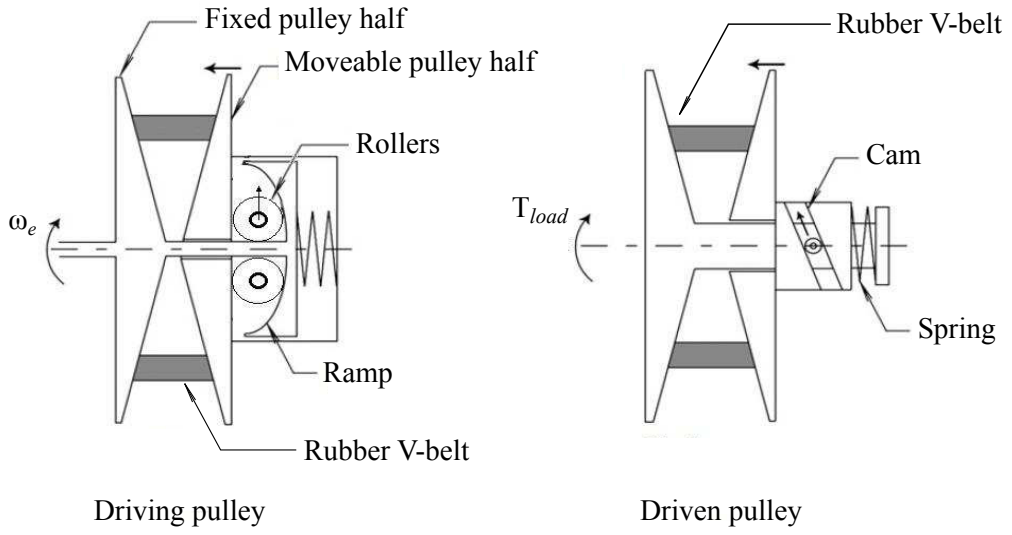



Figure 3.6: Driving pulley (left) and driven pulley (right) of a variable diameter pulley CVT (source: G. Julio et. al [29])

The speed sensing mechanism on the driving pulley utilises centrifugal rollers to actuate axial movement of the pulley halves. As the rollers tend to move outward with the increase in centrifugal force, the pulley halves are forced closer to each other. The torque sensing mechanism on the driven pulley uses a spring-loaded cork screw effect to actuate axial movement of the pulley halves. With the increase of load torque on the driven pulley shaft, the two pulley halves are "screwed on" closer to each other, and when the load decreases the spring forces the two halves to "unscrew" away from each other.

The complex and non-linear nature of this system gives rise to a complicated equation, dependent upon numerous parameters, for prediction of the gear ratio during its operation. Hung and Hong uses bond graphs to derive the dynamics of a similar CVT used in a scooter [25]. Another useful article is by Tseng et al. in 2008 [54], however they use data from a dynamometer test to map engine speeds corresponding to load torque and build a lookup table used in their simulation model.

After various attempts to model the CVT dynamically, it was decided that for the application of this project, the CVT ratio is assumed to be constant. This assumption is based on the implemented limitation of the path planning algorithm that only plans a path to be followed at a constant speed. In Section 3.4.5, an interesting relation between the engine speed and wheel speed relating to the CVT ratio is noted.

### Centrifugal Clutch

The CVT is connected to the clutch, which separates the engine and transmission from the rest of the vehicle drive train when the clutch is disengaged. On the quad bike, a centrifugal clutch is present.

The centrifugal clutch eliminates the need of a manual clutch and ensures the engine does not stall when the quad bike is suddenly slowed or stopped and when the engine is idling. The centrifugal clutch, as indicated by Figure 3.7, consists of an inner disc driven by the output shaft of the CVT, three weighted arms with friction pads and an outer wheel connected to the final drive gears. When the inner disc reaches a certain rotational speed, the centrifugal force swings the weighted arms outward and engages the outer wheel. In the quad bike, the idling speed is $\pm 1700$ rev/min, so the centrifugal clutch would have been designed to engage at a rotational speed above this speed multiplied by the minimum CVT ratio.

For control purposes, the clutch will be assumed to be either fully engaged or fully disengaged. The transitional state when the centrifugal clutch starts to engage and slips will not be modelled, for normal operation of the quad bike will not be near this condition. A combined CVT and clutch slip loss coefficient might be assumed as in the publication by Tseng et.al [54] to account for any slipping that might occur

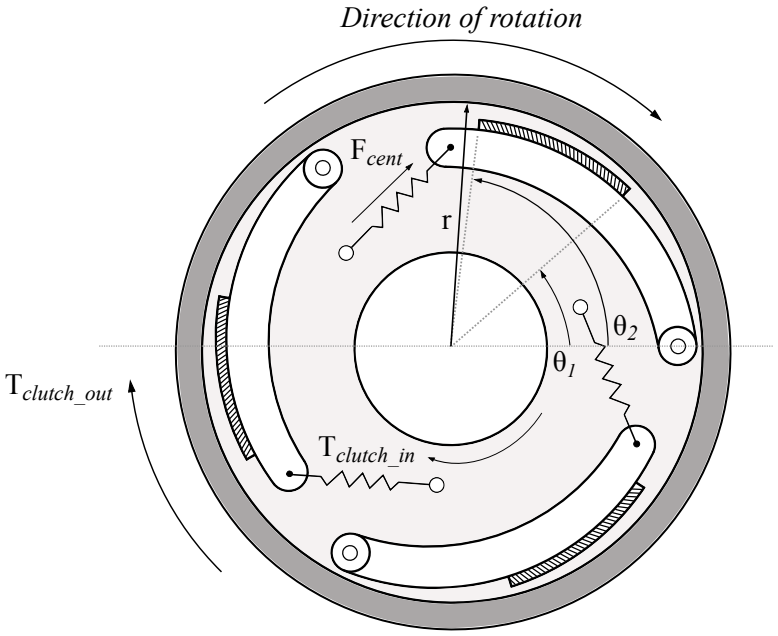



Figure 3.7: Centrifugal Clutch (adapted from *Shigley's Mechanical Engineering Design* [6])

during operation. The books, *Shigley's Mechanical Engineering Design* [6] and *Illustrated Sourcebook of Mechanical Components* [44], explain the workings of and discusses the dynamic modelling of centrifugal clutches, with equations derived from Figure 3.7. These can be used if the simulation model is required to represent the transient effects of the clutch. The following equation describes the model assumed for the centrifugal clutch.

$$T_{clutch\_out} = \begin{cases} 0 & \text{for } N_e \leq 2000 \text{ rev/min} \\ \eta_{slip} T_{clutch\_in} & \text{otherwise} \end{cases}$$

where $T_{clutch\_in}$ is the torque seen by the input shaft of the clutch, $T_{clutch\_out}$ is the transferred torque to the output shaft of the clutch and $N_e$ is the engine speed in rev/min. The combined CVT and clutch slip loss coefficient $\eta_{slip}$ is suggested by Tseng et.al [54] to take the form of an exponential decay function, with a high slip loss during acceleration from low speed and a low slip loss at high speed where minimum acceleration occurs. The remaining symbols in Figure 3.7 are: $r$, the inner radius of the outer wheel of the clutch connected to the final drive, $\theta_1$ and $\theta_2$ which defines the radial length of the friction pads and $F_{cent}$ which is the centrifugal force on the weighted arms that presses the friction pads against the outer wheel during rotation of the inner part of the clutch.

### Final Drive and Chain link

Connected to the output shaft of the clutch is the final drive, followed by the chain link to complete the drive train elements connecting the engine to the rear wheel axle of the

quad bike.

The final drive gears do not only allow the rotational direction of the wheels to change, but also provides additional torque multiplication, which is necessary due to the limited range of the CVT. The final drive consist of a forward gear ratio of $R_{gears\_F} = 7.226$, a reverse gear ratio of $R_{gears\_R} = 26.902$ and a neutral position. The gear selector allows for either the forward gears, reverse gears or neutral to be engaged. For the simulation model these gains will be implemented with an efficiency coefficient $\eta_{gears}$ to compensate for frictional losses in the gears. Take note that the chain drive ratio forms part of the forward and reverse gear ratios given above.

### Disc brakes

A disc brake is a type of brake which slows the rotation of a wheel by the friction caused by pushing brake pads with a set of callipers against a brake disc mounted on the wheel axle. The callipers can be actuated hydraulically, mechanically, pneumatically or electromagnetically.

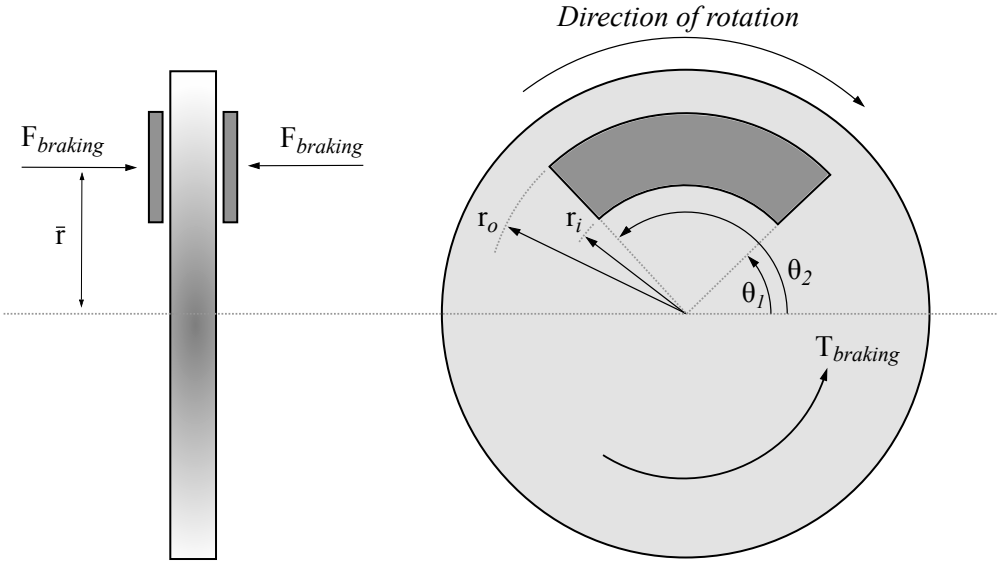



Figure 3.8: Front (left) and side (right) view of the disc brake forces and moments diagram (adapted from *Shigley's Mechanical Engineering Design* [6])

On the quad bike a single disc brake on the rear axle facilitates braking of the rear wheels. The front wheels each have a drum brake, but these are not controllable and are therefore left out of the vehicle drive train modelling. The callipers of the disc brake are actuated by hydraulic pressure, which in turn is automated by the addition of an electric motor to pull the brake lever of the quad bike. The forces on the disc and resulting braking torque on the rear wheel axle, as indicated in Figure 3.8, can be modelled dynamically using *Shigley's*

*Mechanical Engineering Design* [6]. The force gain from the brake lever through the hydraulics to the callipers can be modelled as a constant gain, due to the incompressibility of the brake fluid used. Lastly, the dimensions of the disc brake mechanism can be measured, and is also available in the service manual of the quad bike which can be used as confirmation of the measured values. From *Shigley's Mechanical Engineering Design* [6], the force on the callipers, $F_{braking}$, and the resulting torque, $T_{braking}$, can be given by:

$$F_{braking} = \int_{\theta_1}^{\theta_2} \int_{r_i}^{r_o} pr \, \mathrm{d}r \, \mathrm{d}\theta \quad = (\theta_2 - \theta_1) \int_{r_i}^{r_o} pr \, \mathrm{d}r$$
$$T_{braking} = \int_{\theta_1}^{\theta_2} \int_{r_i}^{r_o} fpr^2 \, \mathrm{d}r \, \mathrm{d}\theta \quad = (\theta_2 - \theta_1) f \int_{r_i}^{r_o} pr^2 \, \mathrm{d}r$$

where $p$ is the local contact pressure between the brake pad and the disc and $f$ is the coefficient of friction. The inner radius of the brake pad is specified by $r_i$ and the outer radius by $r_o$. The effective radius $\bar{r}$ where $F_{braking}$ is applied is approximated for this project to be halfway between $r_i$ and $r_o$, however, can be calculated as presented in *Shigley's Mechanical Engineering Design*. The radial length of the brake pad is given by the difference between the angles $\theta_1$ and $\theta_2$.

The force required to pull the break lever was determined experimentally by Visser [56] when the quad bike was automated. The input command to the brake actuator motor is a normalised value, and the resulting force exerted by the motor is an exponential function between 0 and 500 N (Newtons) corresponding to a signal between 0 and 1. As already mentioned, the relation between the force exerted by the braking actuator and the force on the callipers is a constant gain. This gain can be calculated by comparing the maximum braking force required to bring the quad bike from full speed to a stand still, within the stopping distance specified by the data sheet of the quad bike, and the maximum force exerted by the brake motor. Then, using the force on the callipers and the rotational speed of the wheel axle, the resulting torque applied by braking can be calculated [6].

### Wheels

Detailed dynamic modelling of the pneumatic tyres will not be done for this project, thus the tyres are assumed to be rigid. The motivation for this is that the frequency analysis of the wheels and suspension of the vehicle is of no interest for trajectory tracking. Any error in trajectory following from non-linear behaviour of the tyres will be compensated for by the steering controller. In Chapter 2 of the book, *Motor Vehicle Dynamics: Modelling and Simulation* [19], the principles of dynamic tyre modelling are thoroughly covered and can be used if required.

In Section 3.5 of this thesis the frictional and steering forces on the front tyres are discussed in greater detail, which are used to develop the steering model and steering controllers.

### 3.4.2 Summary of the vehicle speed model

By combining all of the drive train elements from Section 3.4.1 into a unified system model, the vehicle speed plant can be summarised by the simplified diagram shown in Figure 3.9. The controlled inputs to this model will be the throttle position and brakes, the disturbance input will be the road load as seen by the wheels, and the output is the vehicle speed.

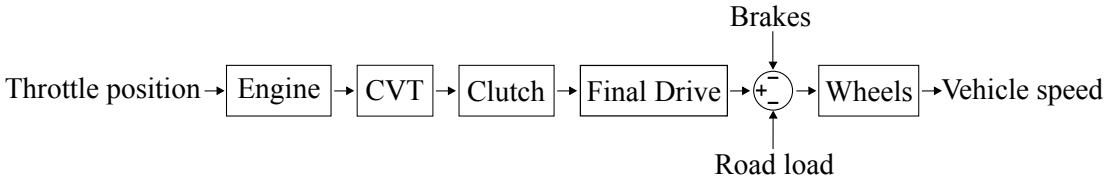



Figure 3.9: Summary of the drive train elements to represent the vehicle speed plant

For a feedback controller to be implemented, measurement values are required. From Figure 3.9, the engine speed, wheel speed and vehicle speed can be measured and are available to the controller.

### 3.4.3 Throttle actuator control

The throttle is actuated by a servo motor, as described in Section 2.2, which is controlled from the on-board controller. During development and testing of the controllers of the quad bike, it was noted that the servo motor behaved irregularly at an engine speed around 6000 rev/min. After measuring the control signals to the servo motor and ensuring this was not the problem, the servo was tested with the engine switched off and then the servo behaved normally. It was deduced that the cause of this irregular behaviour in the servo motor is from the electromagnetic interference from the engine ignition system.

The servo control signal operates at 50 Hz from the on-board controller on the quad bike. This frequency correlates to 3000 times per minute and is exactly half of the engine speed at which the irregular behaviour was noted. The engine used in the quad bike is a single cylinder, 4-stroke engine. By looking at the combustion cycle of a 4-stroke engine, it was found that for every two revolutions of the engine drive shaft, the spark plug ignites the combustion chamber once. This means that when the engine turns at 6000 rev/min, the spark plug fires at 3000 times per minute.

Therefore, when the rate at which electromagnetic pulses are generated by the spark in the combustion chamber of the engine approaches the same rate at which the control signal is sent to the servo motor, interference occurs and the control signal becomes distorted. This interference was eliminated by replacing the control signal cable between the on-board computer and the servo motor with a shielded cable.

For the servo motor that can turn between a normalised position of 0 and 1, the mounted position and throttle cable only allowed the servo motor to actuate between 0.2 and 0.4. This implies that the reference input to the throttle actuator must be limited to this range. Control of the servo is in the form of open-loop control, with no measurement available of the actual position. Therefore, specific servo motor control was not done, and the dynamics of servo motor is assumed to be included in the engine model.

A low pass filter was added to reduce the control effort to the throttle actuator due to measurement noise on the RPM sensor. The design of the low pass filter is discussed below as part of the engine speed controller design.

### 3.4.4 Engine Speed Control

For this project, before it was decided to model the CVT as a constant gear ratio instead of a dynamically changing ratio, an engine speed controller was designed to give the ability to control the input to the speed sensing mechanism of the CVT. The engine speed controller would form an inner loop control of the final vehicle speed controller. Figure 3.10 presents the layout of the controller and plant, where $R(s)$ is the reference engine speed and $Y(s)$ the measured engine speed. Each component in the control layout is discussed in the following subsections.

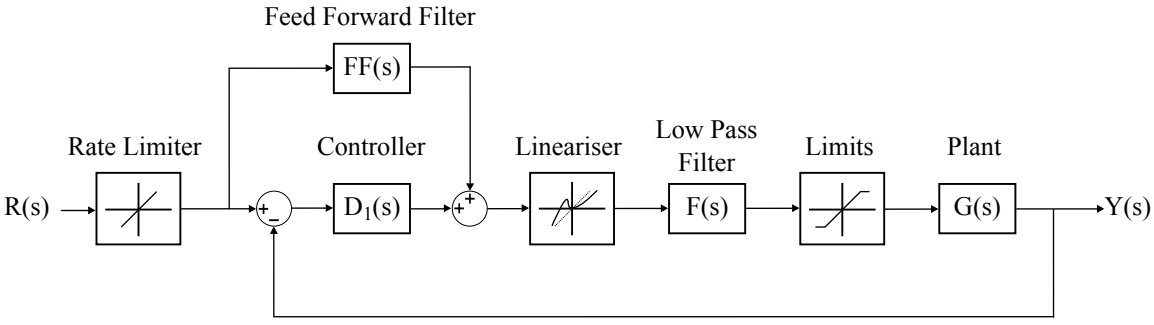



Figure 3.10: Engine speed controller of the quad bike engine

#### Plant

The plant, $G(s)$, considered here is the engine model presented in Section 3.4.1, described by the second-order differential equation. This equation can be expressed in the form of a transfer function from the engine torque to the engine speed

$$G(s) = \frac{\omega_e}{T_e} = \frac{\frac{1}{J_e}}{s + \frac{B_e}{J_e}},$$

where the values for the equivalent engine inertia $J_e$ and viscous friction coefficient $B_e$ were estimated from the test data.

Figure 3.11 illustrates the plant transfer function with the engine speed as the desired output and the engine torque as an input. The equation from the publication by Dragos et al. [13] is used in the simulation model to estimate the engine torque for a percentage of throttle opening at a certain engine speed, but for the purpose of the controller design, this equation is approximated as a constant gain and included into the engine plant model.

From the test data it was also seen that the plant had a time delay of up to 0.12 seconds between a change in throttle command and the measured engine speed response. This can be represented in the Laplace domain as $e^{-ts}$, where $t$ is the time in seconds. The time delay is between the input of the plant and the output of the linearising function of the throttle position to estimated torque model.

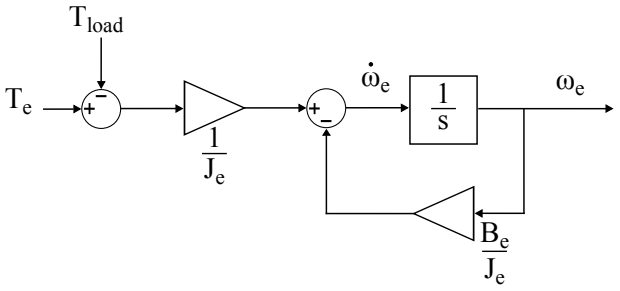



Figure 3.11: Second-order differential plant model of the engine

Shown in Figure 3.11 is $T_{load}$ that represents the load at the wheels as seen by the engine.

### Limits

An upper and a lower limit for the throttle command signal must be applied to prevent mechanical failure in the throttle actuator set-up. As already mentioned in the section on throttle actuator control, the servo motor's motion is bounded between the normalised position values of 0.2 and 0.4.

### Low Pass Filter

The low pass filter mentioned in Section 3.4.3, $F(s)$, takes the form of a first-order low pass filter, with the cut-off frequency chosen to be just on the edge of the bandwidth of the plant model. After simulating the engine speed sensor measurement noise and analysing various test results, a cut-off frequency of 10 rad/s was used and the transfer function of the low pass filter becomes

$$F(s) = \frac{10}{s + 10}.$$

### Lineariser

From Figure 3.5, the equation from the publication by Dragos et al. [13] showed a relatively constant torque for a given throttle position over the engine speed range of a combustion engine, with the throttle position linearly equivalent to the torque being produced up to the maximum torque of the engine. This implies that for a constant increase in throttle position, a constant increase in torque produced would result in the engine plant model producing an engine speed curve similar to an exponential function.

From a test done to measure the engine speed, in a no-load condition with the gear selector in neutral, over the range of the throttle actuator position on the quad bike, it was seen that the engine speed curve was non-linear and not as expected, shown in Figure 3.12. This might be explained by the non-linearities between the throttle actuator and the torque conversion in the engine. The cable connecting the throttle lever, throttle actuator and carburettor was also found to be "sticky" and requiring a non-linear actuation force to be moved.

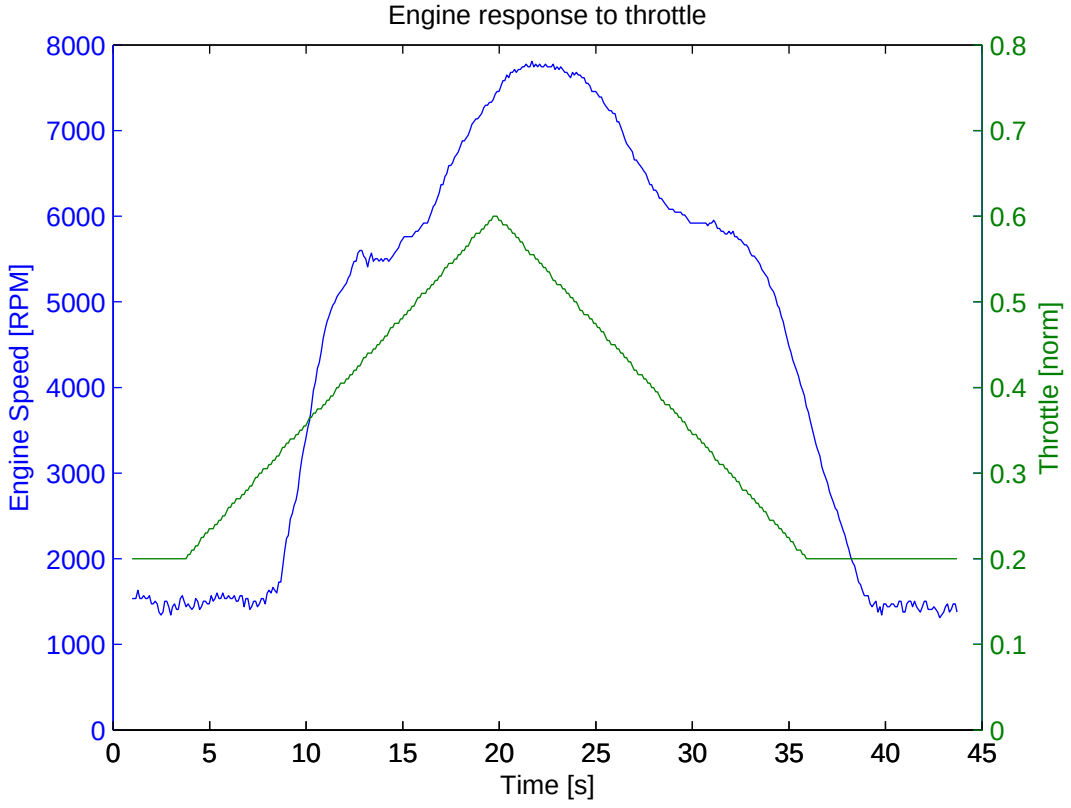



Figure 3.12: Engine speed response to throttle actuator position under no-load conditions

The lineariser approximates the non-linear conversion of the throttle actuator position to estimated torque with straight line segments and calculates what the new throttle actuator

command should be for a desired torque input to the plant model.

### Controller

As mentioned before, the initial aim of the engine speed controller was to be able to control the input to the speed sensing mechanism of the dynamic CVT model. After it became apparent that the CVT could be modelled as a constant gain, it was decided to keep the engine speed controller as the inner loop of the vehicle speed controller. Tracking accuracy of the reference engine speed is not a requirement for this application, therefore, only **proportional** control was added to keep the controller simple and not to add unnecessary dynamics to the system, and can be represented by the transfer function

$$D_1(s) = K_P.$$

By increasing the value $K_P$, the steady state error could be reduced, but with a lightly damped system, the the transient response would be unsatisfactory. By keeping $K_P$ small, the overshoot and settling time becomes acceptable, steady state error is not of importance, but the response time is too slow.

### Feed-Forward Filter

After finding that the proportional controller was not sufficient on its own, it was decided to add a feed-forward controller, $FF(s)$, to improve on the response time.

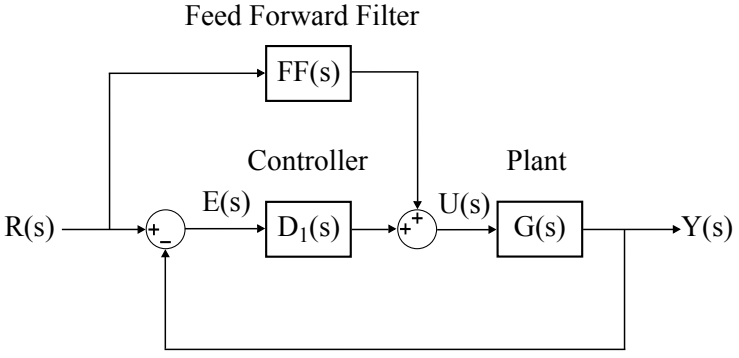



Figure 3.13: Simplified plant and controller with feed-forward filter

From Figure 3.13, the form of $FF(s)$ can be derived from the closed loop equation starting from the output with

$$Y(s) = G(s)U(s)$$

and substituting $U(s) = FF(s)R(s) + D_1(s)E(s)$ the output can be written as

$$Y(s) = G(s)\left[FF(s)R(s) + D_1(s)E(s)\right].$$

Now, by replacing $E(s) = R(s) - Y(s)$, we get

$$\begin{aligned} Y(s) &= G(s)\left[FF(s)R(s) + D_1(s)\left(R(s) - Y(s)\right)\right] \\ Y(s) &= G(s)\left(FF(s) + D_1(s)\right)R(s) - G(s)D_1(s)Y(s) \\ Y(s) + G(s)D_1(s)Y(s) &= G(s)\left(FF(s) + D_1(s)\right)R(s) \\ \left(1 + G(s)D_1(s)\right)Y(s) &= \left(G(s)FF(s) + G(s)D_1(s)\right)R(s) \\ \frac{Y(s)}{R(s)} &= \frac{G(s)FF(s) + G(s)D_1(s)}{1 + G(s)D_1(s)}. \end{aligned}$$

We want the output of the system to be equivalent to the input: $Y(s) \equiv R(s)$, therefore the ideal feed-forward filter is calculated as

$$\begin{aligned} G(s)FF(s) + G(s)D_1(s) &= 1 + G(s)D_1(s) \\ G(s)FF(s) &= 1 \\ FF(s) &= \frac{1}{G(s)}. \end{aligned}$$

It can be seen that the feed-forward filter takes the form of the inverse of the plant; however, this is not realisable, since there are more plant poles than plant zeros and $FF(s)$ should have more zeros than poles. It was decided to simplify the feed-forward filter as a linear gain.

### Rate Limiter

The rate limiter was implemented to limit the maximum rate of change of the reference engine speed input command to especially prevent sudden acceleration of the engine that could result in a forward jerk of the vehicle. This is also a safety precaution for the human driver that would have to be present on the vehicle during autonomous driving. The reference command was limited to increase or decrease at a constant rate of 1000 rev/min over a period of 1 second.

### Resulting engine speed control field test results

The controller described above was implemented on the OBC of the test vehicle and Figure 3.14 presents the field test results. The engine speed reference was given via the drive-by-wire controller to the OBC by the human driver on the quad bike while driving. The vehicle was in a stationary position, accelerated forward by the controller with an engine speed reference, and then decelerated to an idling state. In Figure 3.14, the black curve represents the measured engine speed while driving and the red curve represents the reference engine speed command. The reference engine speed command was then put through the simulation model of the engine speed controller and plant to produce the blue

curve. This gives the ability to visually compare the simulated model with the actual model. It should be noted that the disturbances were not known to the simulation model, which also explains part of the difference.

To summarise, accurate engine speed control is not the desired outcome, but rather the acceptable control of the vehicle speed for the purpose of manoeuvre control, as discussed in more detail in Chapter 4. Therefore, the performance indicated in Figure 3.14 of the engine speed controller was deemed sufficient. The vehicle speed controller is discussed in the following section.

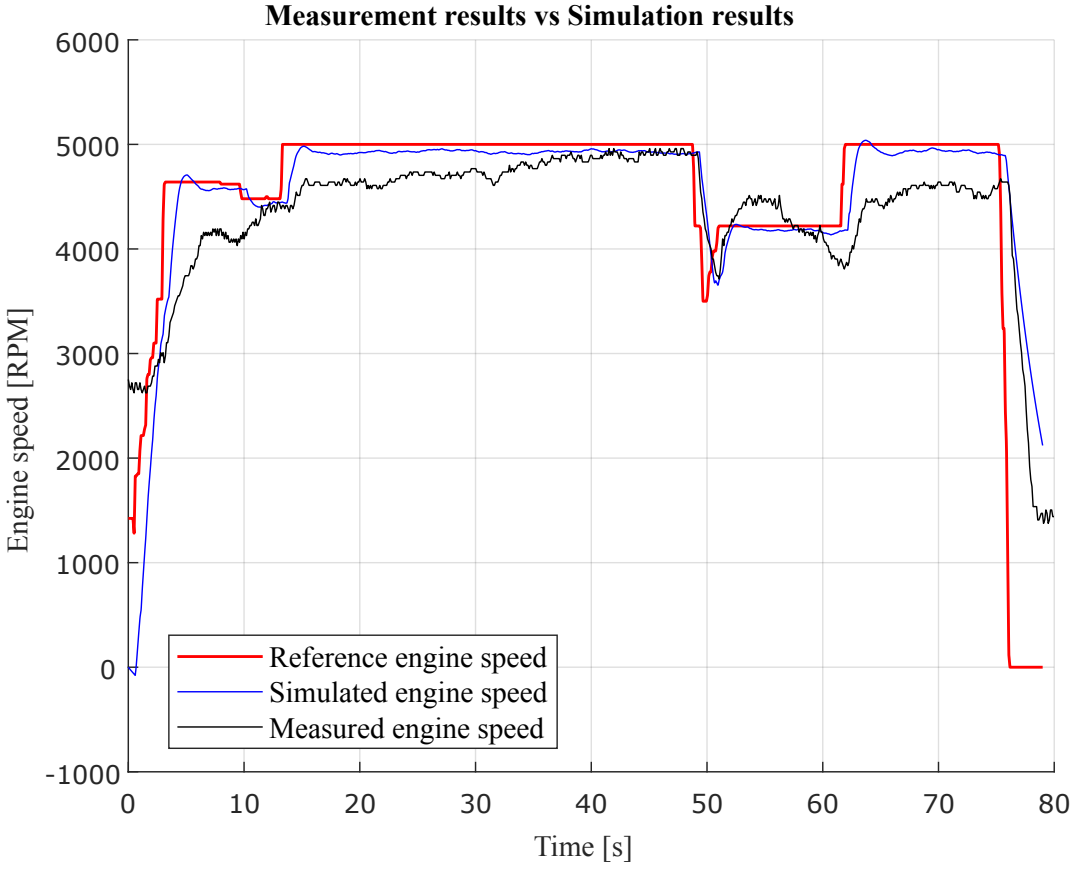



Figure 3.14: Comparing the simulated engine speed model and controller with field test results

### 3.4.5 Wheel Speed Control

By following the consecutive loop closure control approach, the vehicle speed controller $D_2(s)$ closes the control loop around the engine speed controller $D_1(s)$, engine model $G(s)$ and drive train gains $K$, as indicated in Figure 3.15. For the purpose of the vehicle speed controller design, it was assumed that the CVT and other drive train elements can be

modelled as a constant gain to produce the wheel speed, and implicitly the vehicle speed, when no slipping between the wheels and terrain occur.

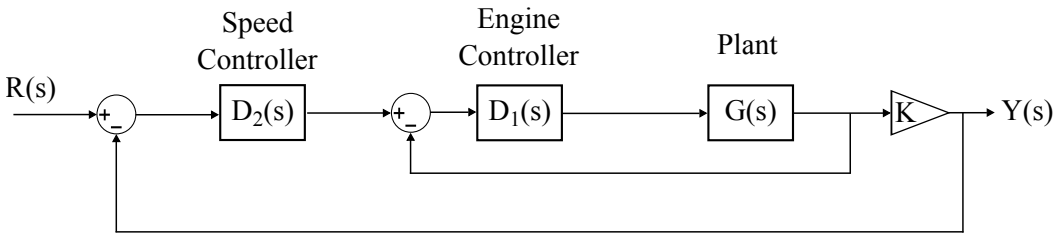



Figure 3.15: Simplified closed-loop vehicle speed control

### Controller

For the vehicle speed control, a proportional plus integral (PI) controller was chosen to add the ability to reduce the steady-state tracking error to zero. The controller takes the form of

$$D_2(s) = K_P + \frac{K_I}{s}$$

in the Laplace domain, where $K_P$ is the proportional gain term, $K_I$ the integral gain and $s$ the Laplace integrator.

### Resulting vehicle speed control field test results

The vehicle speed controller was implemented on the OBC and the designed values for $K_P$ and $K_I$ were adjusted during field tests until a satisfactory performance was achieved, presented in Figures 3.16 and 3.17.

Figure 3.16 shows the measured, reference and simulated values of the inner control loop, that is the engine speed controller. Figure 3.17 shows the measured, reference and simulated values of the outer control loop, that is the vehicle speed controller.

From Figure 3.17 it can be seen that the controller's best performance is around a 3 m/s reference vehicle speed, where the steady-state tracking error is within a 0.5 m/s error band. This would become the constant speed reference for the path planning algorithm when finding and producing a path to follow. The large offset for the reference wheel speed of 0.5 m/s is due to the vehicle controller state machine that provides a reference engine speed of 3800 rev/min for any wheel speed reference value below a specified threshold. For this test the state machine was set to only change over to wheel speed control for reference values from 2 m/s and above. This was done to prevent the wheels speed controller from trying to control the vehicle speed at a reference where extensive clutch slip might occur.

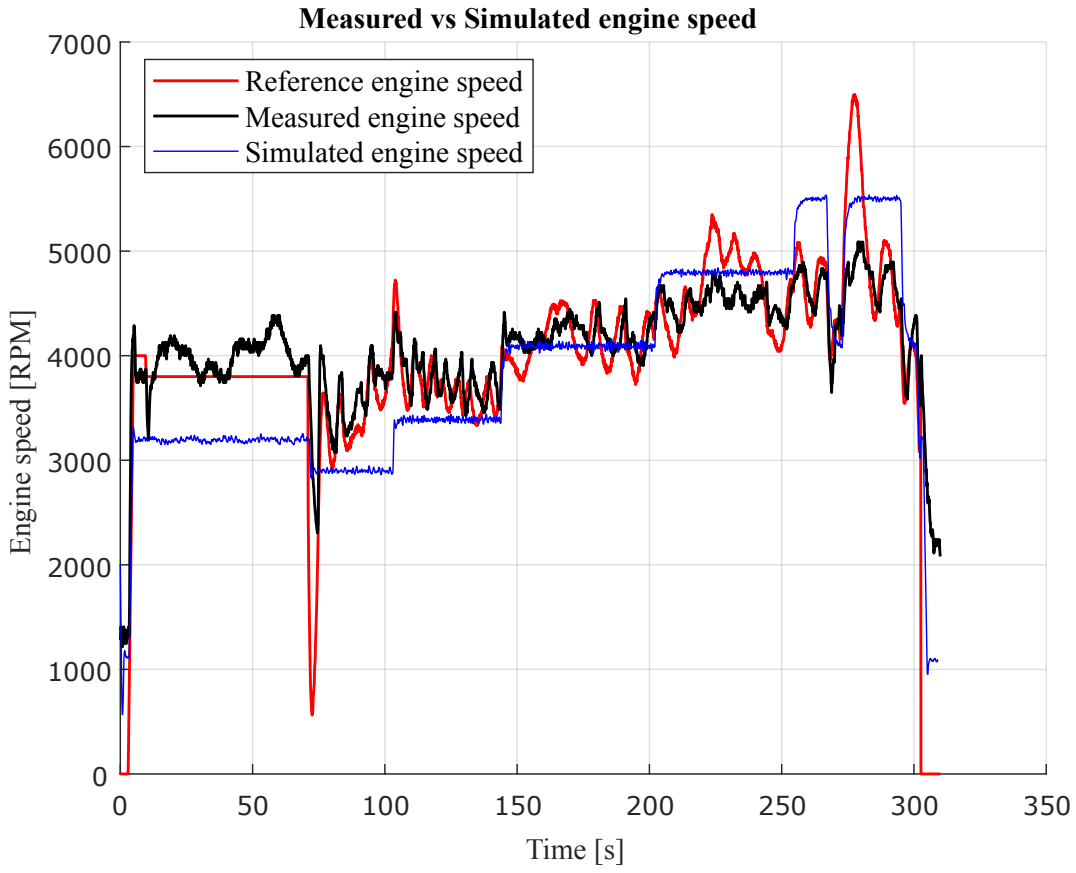



Figure 3.16: Comparing the simulated engine speed model and controller with field test results

The oscillating wheel speed seen in Figure 3.17 might be explained by the unmodelled effect of the CVT and gear ratios between the engine and wheels assumed for the engine model. The derivation of this effect follows.

For the second-order equation of the engine model

$$J_e\dot{\omega}_e + B_e\omega_e = T_e - T_{load},$$

where $T_{load}$ is the torque at the wheels $T_w$ seen by the engine, that is the torque at the wheels divided by the gear and CVT ratio $R_K$

$$T_{load} = (T_w)_e = \frac{T_w}{R_K}.$$

Note that, from the engine to the wheels, the CVT and gears act as a torque multiplier and a speed divider, which means that

$$\frac{T_w}{T_e} = R_K \quad , \quad \frac{\omega_w}{\omega_e} = \frac{1}{R_K} \quad \text{and} \quad \frac{\dot{\omega}_w}{\dot{\omega}_e} \approx \frac{1}{R_K}.$$

The torque at the wheels can be represented by the accelerating force $F_{vehicle}$ of the vehicle multiplied by the radius of the wheel $r_w$

$$T_w = F_{vehicle}r_w = m_{vehicle}a_{vehicle}r_w.$$

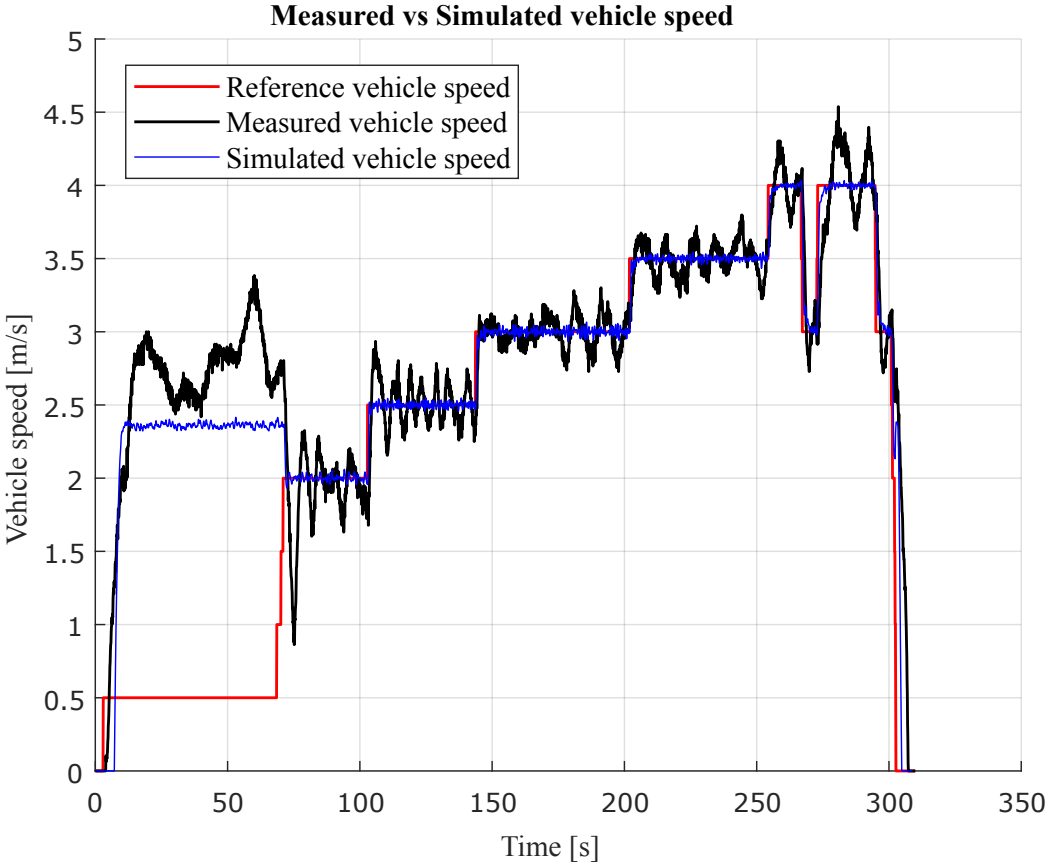



Figure 3.17: Comparing the simulated vehicle speed model and controller with field test results

The acceleration of the vehicle $a_{vehicle}$ is equivalent to the acceleration of the wheel $\dot{\omega}_w$ multiplied by the radius of the wheel, and the equation above becomes

$$T_w = F_{vehicle} r_w = m_{vehicle} \dot{\omega}_w r_w^2.$$

Now, by considering the load torque again, it can be represented by

$$T_{load} = (T_w)_e = \frac{T_w}{R_K} = \frac{m_{vehicle} \dot{\omega}_w r_w^2}{R_K}$$

and by substituting $\frac{\dot{\omega}_w}{\dot{\omega}_e} \approx \frac{1}{R_K}$ we get

$$T_{load} \approx \frac{m_{vehicle} \dot{\omega}_e r_w^2}{R_K^2}$$

By substituting the approximation of the load torque back into the second-order equation of the engine,

$$J_e \dot{\omega}_e + B_e \omega_e = T_e - \frac{m_{vehicle} r_w^2}{R_K^2} \dot{\omega}_e$$

and rearranging it to, we get

$$\begin{aligned} T_e &= J_e\dot{\omega}_e + \frac{m_{vehicle}r_w^2}{R_K^2}\dot{\omega}_e + B_e\omega_e \\ &= \left(J_e + \frac{m_{vehicle}r_w^2}{R_K^2}\right)\dot{\omega}_e + B_e\omega_e. \end{aligned}$$

Disregarding the steady state term and rewriting the equation in a transfer function form to represent it as a gain, we get

$$\frac{\dot{\omega}_e}{T_e} = \frac{1}{J_e + \frac{m_{vehicle}r_w^2}{R_K^2}}.$$

Lastly, the engine inertia $J_e$ is assumed to be very small compared to the vehicle inertia, i.e. $J_e \ll \frac{m_{vehicle}r_w^2}{R_K^2}$, and the gain becomes

$$\frac{\dot{\omega}_e}{T_e} = \frac{R_K^2}{m_{vehicle}r_w^2}$$

From this derivation, it can be seen that the engine and drive train model includes a quadratic term, and the effect becomes propagated in the control loop for a non-constant CVT ratio. However, from Figure 3.17, it can be seen that the oscillations are relatively small and slow. It is less than 0.5 m/s peak-to-peak over a period of approximately 7 seconds, and was barely noticeable to the human driver present on the quad bike during the test. It was therefore decided that the vehicle speed controller is sufficient for executing a planned path autonomously.

## 3.5 Orientation Control

Now that it is possible to control the driving speed of the vehicle autonomously, the control of the direction in which the vehicle has to drive is still outstanding. This section discusses the steering mechanism control of the quad bike, as well as the heading controller, and concludes with the addition of a cross-track error controller to reduce small deviations from the path calculated by the path planner.

### 3.5.1 Steering Angle Actuator Control

Figure 3.18 presents a visualisation of the description in Section 2.2 of the steering actuator and mechanism implemented by a previous project to automate the steering of the vehicle. Starting from the left, the electric representation of the electric steering actuator is given, followed by the worm gearbox on the motor. On the gearbox output shaft a small sprocket is attached, connected to a larger sprocket with a chain link. The larger sprocket is attached to the steering link. From Figure 3.18, $T_m$ and $\omega_m$ is the torque and angular velocity of

the motor shaft, $T_G$ and $\omega_G$ the torque and angular velocity of the gearbox shaft, $T_C$ and $\omega_C$ the torque and angular velocity of the larger sprocket. $T_{WL}$ is the resulting actuating torque on the left wheel steering link, and $T_{WR}$ on the right wheel steering link.

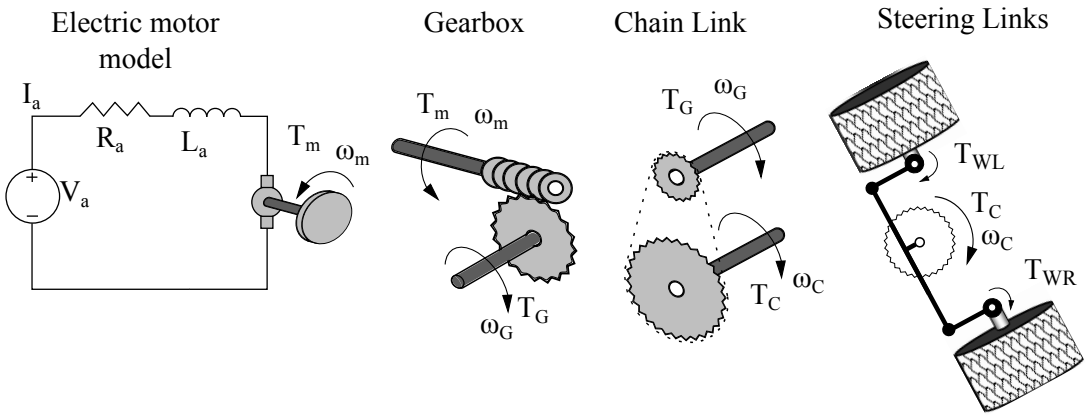



Figure 3.18: Schematic of the quad bike steering actuator and steering mechanism

The steering actuator plant and controller was used from a concurrent project on the quad bike. The electric motor actuating the steering was modelled with electrical and mechanical dynamics, and is represented by the plant $G(s)$ along with the integrator $\frac{1}{s}$ in Figure 3.19. The speed of the actuator is controlled by the inner loop controller $D_2(s)$ and the position of the steering actuator is controlled by the outer loop controller $D_1(s)$. The reference steering angle position is represented by $\theta_{ref}(s)$, the measured angular speed by $\dot{\theta}$ and the measured angular position by $\theta$.

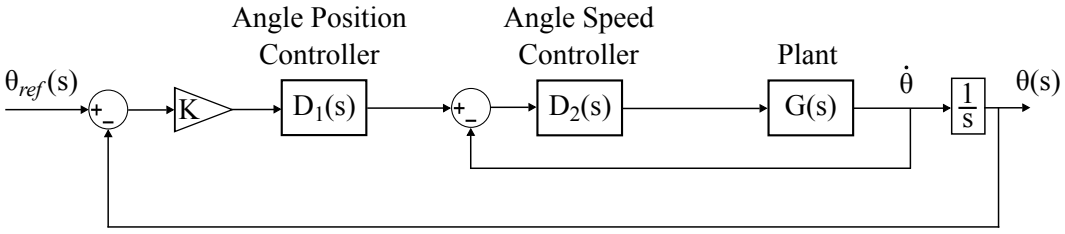



Figure 3.19: Steering actuator control model

The controller design was done in the concurrent project by only modelling the actuator, and did not include the rest of the steering mechanism or attempt to model the load on the front wheels while driving. Field results for the given steering controller, as seen in Figure 3.20, was slow with not enough control energy to turn the steering mechanism close to the specified reference steering angle, and resulted in large turning circles.

For this project, a faster steering controller is required that can produce an automated vehicle turning circle closer to what is producible by a human driver. To achieve improved performance of the controller, a better understanding of the load on the steering actuator and mechanism behaviour should be acquired. By considering the contact area between

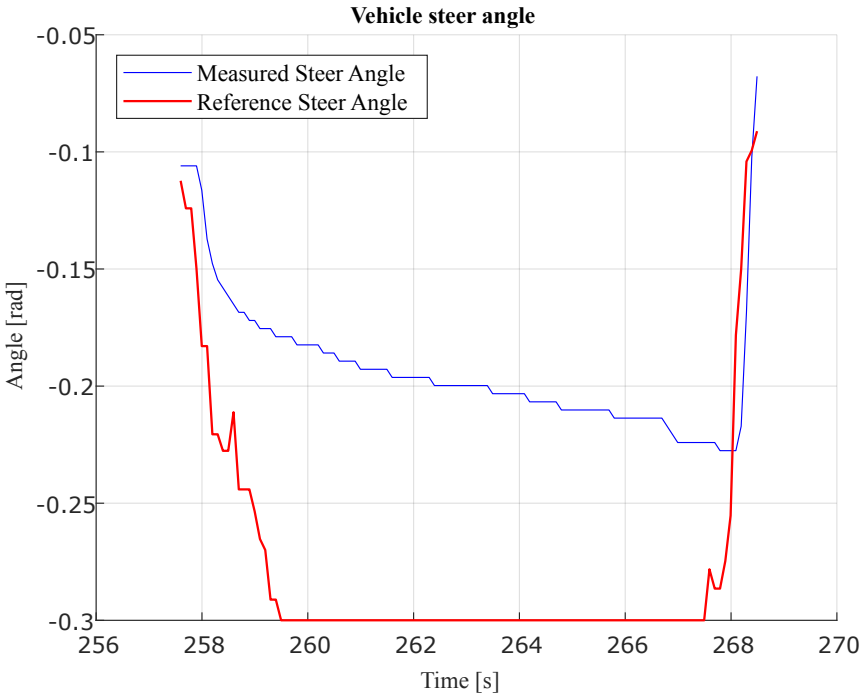



Figure 3.20: Right turn steering angle control before any improvements

the front wheels and the terrain, the frictional force and resulting load torque can be derived from Figure 3.21. The friction coefficient is dependent on the terrain as well as the driving speed at which turning occurs. For this project a constant driving speed will be used for the path planner, and an average coefficient for different terrains will be assumed to model the load, instead of adapting the controller for specific terrains.

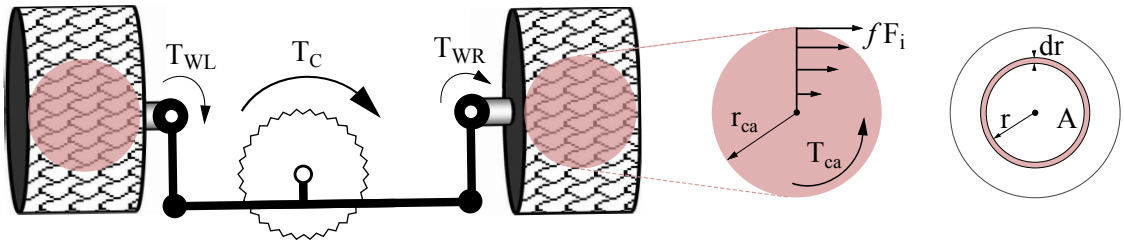



Figure 3.21: Frictional load between the front wheels and the terrain when steering

In Figure 3.21, it is assumed that the contact area $A_{ca}$ beneath each wheel can be represented by a uniform circle with radius $r_{ca}$ and the pivot point when turning the wheels lies in the centre of this circle. The frictional force is distributed along the radius of the contact area, and is a product of the friction coefficient $f$. A uniform pressure $p_a$ is assumed over the total contact area due to the weight on the wheel, and can be written as

$$p_a = \frac{F}{A_{(ca)}} = \frac{\frac{1}{2}m_{front}g}{\pi r_{ca}^2},$$

where the force $F$ on one of the front wheels is half of the total mass on the front part of the vehicle, from Table 2.1. The torque $T_{ca}$ is found by integrating the product of the frictional force $F_i$ and radius of the contact area along the radius of the contact area

$$T_{ca} = \int f F_i r.$$

Substituting $F_i$ with the pressure $p_a$ over the area element $A$

$$T_{ca} = \int f A p_a r,$$

where A can be given by the incremental area $dA(r) = 2\pi r \mathrm{d}r$ in Figure 3.21 to get

$$T_{ca} = \int f 2\pi r \mathrm{d}r p_a r$$

and rearranging

$$T_{ca} = \int 2\pi r^2 f p_a \mathrm{d}r = 2\pi f p_a \int r^2 \mathrm{d}r.$$

Integrating the above equation produces

$$T_{ca} = \frac{2}{3}\pi f p_a r_{ca}^3.$$

The final form of the load, on one of the front wheels, can be found by substituting the uniform pressure on the total contact area

$$T_{ca} = \frac{2}{3}\pi f \left( \frac{\frac{1}{2} m_{front} g}{\pi r_{ca}^2} \right) r_{ca}^3 = \frac{1}{3} f m_{front} g r_{ca}.$$

Another load torque on the front wheels is the self-aligning torque. This torque is responsible for the effect of the steering mechanism that tends to return back to a zero steering angle from a given steering angle, while driving. Figure 3.22 illustrates the effect of the self-aligning torque on one half of the steering mechanism and front wheel. At the contact area between the wheel and the terrain, while driving the tyre deforms slightly to produce a lagging part that is trailing the centre of the wheel by length $t$. When the front wheels are turned, this lagging part has a resulting force vector (red arrow) that produces the self-aligning torque on the steering mechanism.

From Figure 3.22, $t$ is called the pneumatic trail of the tyre, $d$ is referred to as the scrub length and the self-aligning torque $T_{align}$ can be described by

$$T_{align} = F_{lat,front} \sqrt{t^2 + d^2}$$

where $F_{lat,front}$ is given in Section 3.3.2 and is a function of the vehicle speed and steering angle.

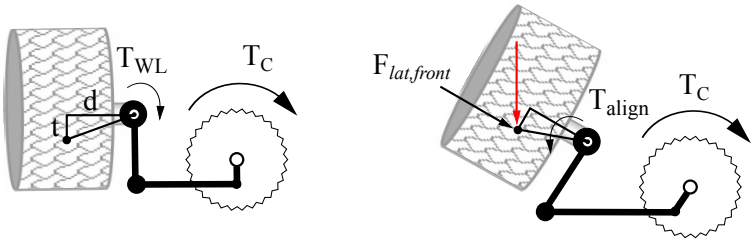



Figure 3.22: Self-aligning effect on the steering mechanism

During the testing of the steering actuator and mechanism, some non-linearities in the steering mechanism were noted. The first non-linearity was backlash on the actuator from to the chain link connecting the actuator and steering mechanism. The chain link is a non-rigid link and allows for a degree of play between the rotational translation of the small and large sprockets. The implication of this effect on the control system is that whenever the actuator changes direction of rotation, there is a short instant where the steering load is disconnected from the actuator.

Another non-linearity was the non-symmetrical performance of the steering actuator. It was noted that during field tests, the vehicle turned sharper to one side than to the other. The actuator itself was suspected to be turning faster in one direction than in the other direction, and was confirmed by removing the actuator from the vehicle and testing it separately. By limiting the current to the actuator, the voltage polarity was applied first in one direction, and the time to rotate the shaft of the actuator once was measured. This was repeated with the voltage polarity reversed. It was deducted that the electric motor was more efficient at converting electrical energy into mechanical energy in one direction of rotation. The original application of the electric motor was to rotate only in one direction, and might have been designed for this purpose only to save on manufacturing costs.

The efficiency problem of the actuator was overcome by increasing the current limitation on the direction of rotation that was slower. By allowing the angular speed control loop controller $D_2(s)$ to provide a higher current reference to the slower direction of rotation, the actuator performance was improved. For safety, a current monitoring function was implemented to prevent overheating of the actuator during prolonged high-current operation.

Power consumed by the actuator is equivalent to the heat generated,

$$\text{Heat} \equiv \text{Power} = I^2 R.$$

Looking at a time window $t$, the safe maximum current $I_{safe}$ for that period $t$ will result in the maximum power (heat) which can be dissipated safely without overheating

$$\begin{aligned} Pt = \bar{I}^2 Rt &\leq I_{safe}^2 Rt \\ \bar{I}^2 &\leq I_{safe}^2 \quad \text{for time window } t. \end{aligned}$$

By having a function on the actuator controller (ACB) to measure the running average of the current $\bar{I}$ over the time window $t$, whenever $\bar{I}^2$ becomes larger than $\bar{I}^2_{safe}$, an alarm can be signalled to the autonomous navigation system to warn against equipment damage.

The estimated load $T_{ca}$ and $T_{align}$ on the steering mechanism and actuator were added to the simulation model and the steering controller's response was improved by only increasing the gain $K$, from Figure 3.19, and keeping the rest of the controllers, $D_1(s)$ and $D_2(s)$, as they were. This resulted in faster and more accurate steering angle control, but large and fast oscillations under no-load conditions were noted. Very small 'no-load' conditions occur when the actuator changes direction of rotation and the effect of backlash is witnessed. This became very prominent at a zero reference steering angle, that is when driving in a straight line. The steering actuator oscillated sharply between a small left and right steering angle and jerked the vehicle slightly to the left and right. The effect on the trajectory of the vehicle with these steering oscillations were minimal, but to prevent damage to the actuator and mechanism, this was unacceptable. The oscillations were eliminated by making the gain $K$ variable, and proportional to the absolute offset of the reference command $\theta_{ref}$ to zero. This implies that at a zero angle reference command, the gain $K$ is the same as it was used in the concurrent project, and the system is stable, but increases proportionately to the absolute magnitude of the reference command, resulting in the system to respond faster in reaching the desired steering angle. Figure 3.23 plots the performance of the steering controller, for a right turn section during a field test done, to show the increase in performance with the improvements done to this point.

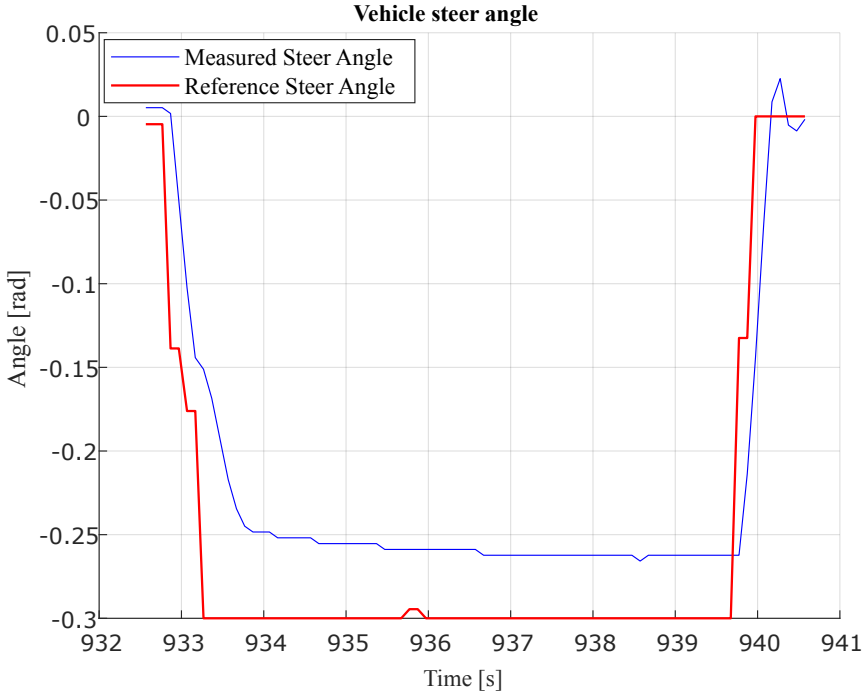



Figure 3.23: Right turn steering angle control with increased controller gain

### 3.5.2 Steering feed-forward Control

For this project, autonomous navigation is the desired outcome. The path planner will calculate a path for the vehicle controller to execute. The path planner uses manoeuvres to find a path, and adheres to the kinetic and dynamic constraints of the vehicle model. Therefore, a very good estimate of the vehicle states for a specific trajectory or manoeuvre are already calculated before the vehicle controller executes the trajectory or manoeuvre. This means that for a given path section received from the path planner, the steering angles at each interval are known, i.e. the output of the steering plant $\theta(s)$ from Figure 3.19. The steering plant has also been modelled, and therefore the input to the plant can be calculated and used as a feed-forward signal in addition to the controller output. In an attempt to further increase the performance of the steering controller, this feed-forward signal is implemented as shown in Figure 3.24.

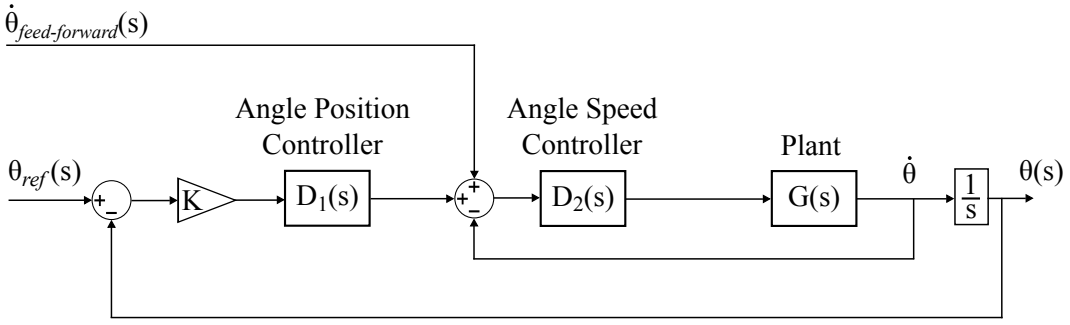



Figure 3.24: Steering controller with a feed-forward signal

At first the calculation was approached as described in Section 3.4.4 where the feed-forward filter of the engine speed controller takes the form of the inverse of the plant. By taking the inverse of the steering plant, in this case the combination of $D_2(s)$, $G(s)$, and $\frac{1}{s}$, it contains unnecessary high frequency elements. The plant is therefore approximated as a plain integrator $\frac{1}{s}$.

This implies that the feed-forward signal $\dot{\theta}_{feed-forward}$ can simply be calculated by taking the derivative of $\theta(s)$ for the given path length.

When calculating $\dot{\theta}_{feed-forward}$, it is not necessary to know the system, only the limitations of the system. The limitations of the steering system is the maximum steering angle and rate of change of the steering angle. The calculated feed-forward signal can even be adapted for different types of terrain if the system's limitations for the respective terrain is known.

Figure 3.25 provides a field result plot of the measured steering angle and reference steering angle with the feed-forward signal implemented by using a right turn manoeuvre control. It can be seen that the steering angle tracking of the controller has improved from Figure 3.23 to Figure 3.25 with a faster response time and almost zero steady-state tracking error.

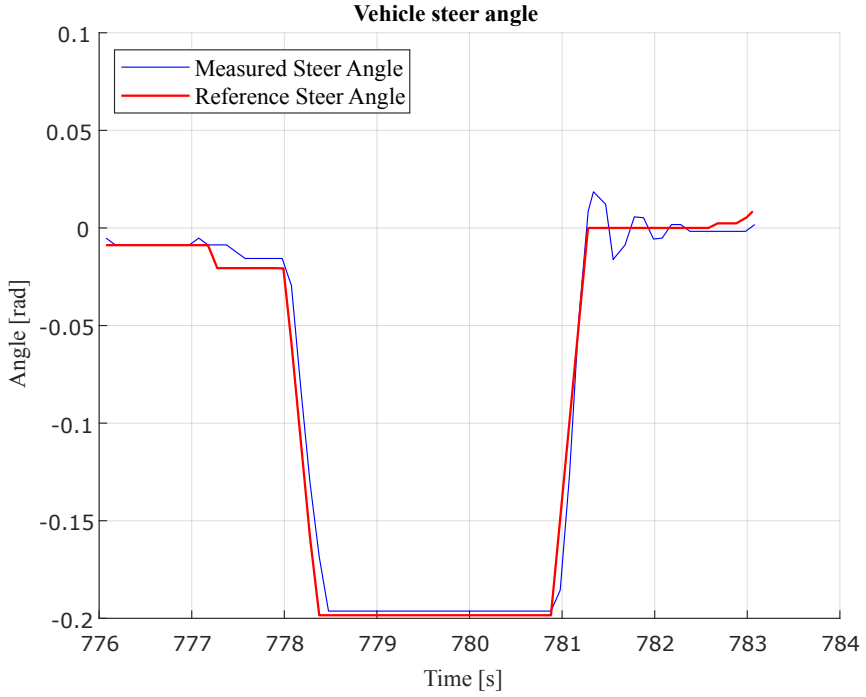



Figure 3.25: Right turn steering angle control with feed-forward manoeuvre control

### 3.5.3 Cross-track Error Control

The last component required to achieve autonomous orientation control of the vehicle is the cross-track error calculator and controller, which results in a guidance controller capable of controlling the vehicle to follow a given path provided by the path planned algorithm.

The cross-track error calculation method implemented in this project is adapted from the guidance controller used on the other AutoNav vehicles. The guidance controller uses straight line segments between predefined waypoints to define the ground track, and then controls the vehicle's heading to reduce the position error from the ground track to zero.

From Figure 3.26, the track heading $\Psi_{track}$ and ground track length $L_{track}$ between two consecutive waypoints, source $(E_{src}, N_{src})$ and destination $(E_{dest}, N_{dest})$, can be calculated with

$$\tan \Psi_{track} = \frac{E_{dest} - E_{src}}{N_{dest} - N_{src}} \qquad \text{and}$$
$$L_{track} = \sqrt{(N_{dest} - N_{src})^2 + (E_{dest} - E_{src})^2}.$$

The vehicle's cross track error $y_{error}$ is the perpendicular distance between the ground track and the vehicle's position, and the in-track distance $x_{track}$ is the projected distance of the vehicle along the track length from the source waypoint.

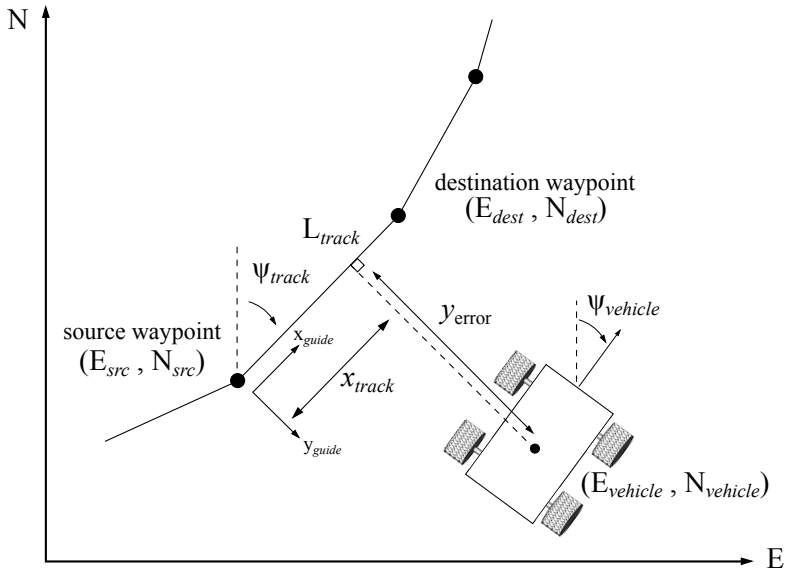



Figure 3.26: Defining the system for cross-track error calculation

Furthermore, to describe the equations used in the cross-track error calculations, a guidance axis system, $x_{guide}y_{guide}$-axis, is defined with its origin at the source waypoint and rotated by the track heading $\Psi_{track}$. The vehicle's position in the NED-axis system is then rotated and translated to the guidance axis system, with its $(x, y)$ position becoming

$$x = (N_{vehicle} - N_{src}) \cos \Psi_{track} + (E_{vehicle} - E_{src}) \sin \Psi_{track}$$
$$y = -(N_{vehicle} - N_{src}) \sin \Psi_{track} + (E_{vehicle} - E_{src}) \cos \Psi_{track}.$$

Note that the guidance axis system is translated and rotated for each source waypoint received from a scheduler, and is valid for the duration or length of $L_{track}$ of the source-destination track.

In this project, the path planner calculates a trajectory of the vehicle containing vehicle and controller reference state information at intervals corresponding to the sampling rate of the vehicle control system. It is therefore possible to use this information as "waypoints" in the calculation of the cross-track error.

The cross-track error control command is added to the steering controller of the vehicle as illustrated by Figure 3.27. For low vehicle speeds, it can be assumed that the side slip angle is equal to the steering angle, and therefore the rate of change in the vehicle heading will be proportional to the steering angle. The rate of change in the heading will change proportional to the vehicle speed as well.

At first, a PI-controller was considered for the cross-track error controller. This would, however, not work because the steering system can be approximated as an integrator $\frac{1}{s}$. By evaluating the Bode-analysis of the controller and plant, for high gains, the -180 degree phase shift would be reached.

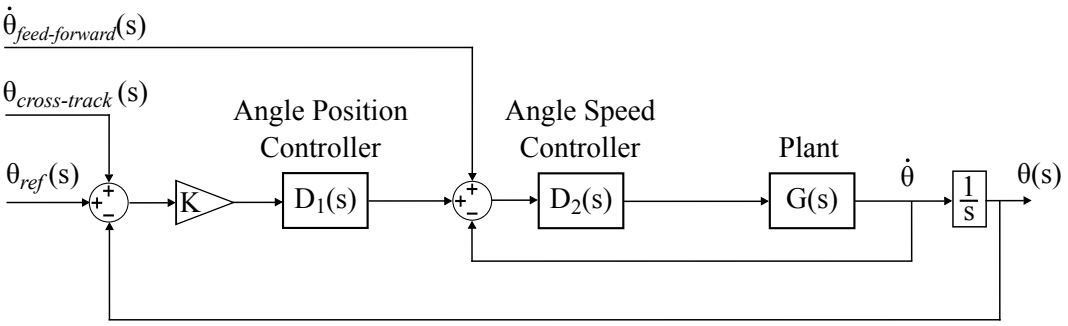



Figure 3.27: Cross-track control added to the steering control system

If a PD-controller was added, the controller bandwidth in the Bode-analysis could be increased, but high frequency effects would become prominent. Sudden changes in the heading of the vehicle is not a desirable outcome.

With a lead-compensator, the response time of the system could be increased at the cost of an offset error. A lag-compensator would increase the tracking accuracy of the system, at the cost of response time.

To keep the cross-track error controller simple, the assumption is made that when driving in a straight line, the steering angle $\theta$ is proportional to the cross-track error $y_{error}$. Therefore, the cross-track error calculator can be approximated as a linear gain. This implies that only the dynamics of the steering system has to be considered to design the cross-track error controller, and not the full lateral dynamics of the vehicle as described in Section 3.3.2.

After simulations and design, the cross-track error controller implemented in this project took the form of a lead-compensator and is represented by the transfer function

$$D_{crosstrack}(s) = K\frac{s+z}{s+p}$$

where constant $z < p$ and $K$ is the proportional gain.

The error calculation is implemented as

$$y_{error} = (E_{vehicle} - E_{src})\cos\Psi_{track} - (N_{vehicle} - N_{src})\sin\Psi_{track}$$

and is added to the steering control loop in Figure 3.27 with

$$\theta_{cross-track}(s) = y_{error}(s)D_{crosstrack}(s).$$

The performance of the cross-track error controller will be evaluated in Chapter 8 on the field test results done with a path generated by the path planner algorithm.

This concludes the chapter on the vehicle controllers. Now that the vehicle can be controlled and the capabilities of the vehicle is known, a set of repeatable trajectories that the vehicle can execute can be compiled. This set of trajectories, or manoeuvres, is the subject discussed in the next chapter.

# Chapter 4

# Manoeuvres and the Local Planning Method

Planned paths are built up from manoeuvres, and therefore the chapter starts by clarifying the concept of manoeuvres, followed by the previous implementation thereof on the test vehicle. Improvements on the steering model are discussed and how they influence the manoeuvres. The local planning method is the part of the path planner that strings manoeuvres together. The rest of the path planner is discussed in Chapter 6.

## 4.1 Manoeuvres

### 4.1.1 Concept of Manoeuvres

From the discussion of the autonomous navigation architecture in Section 1.2, it was briefly mentioned that a manoeuvre is a set of movements used to move the vehicle between two points. More specifically, a manoeuvre is a quantisation of system dynamics and represented by finite time parametrised curves, as discussed in length in the publications by Frazzoli et al. in [15, 17, 16]. Ultimately, the use of manoeuvres reduces the computational complexity of the path planner for non-linear or high dimensional systems by restricting the feasible trajectories to a family of nominal, executable trajectories.

With the vehicle controllers in place and tested, the capabilities of the vehicle are known. Using this information, a list of predetermined trajectories that the vehicle can execute can be compiled. This list of trajectories, or movements, is called the manoeuvre library. By using the manoeuvre library, it can be ensured that the paths created by using manoeuvres will also be executable. The following section discusses the manoeuvre library used in previous work on the test vehicle and further development made for this project.

### 4.1.2 Previously Implemented Manoeuvres

Previous work done by Visser [56] was based on a simple car, as described by the Dubins model. For this model only forward driving is considered and the path of the vehicle can

be described by straight lines and constant curvature circles. The resulting manoeuvres used (from a stationary position) are as follows:

- Turn left. Drive in an arc. Stop. Turn straight. Drive straight. Stop.
- Turn right. Drive in an arc. Stop. Turn straight. Drive straight. Stop.
- Drive straight. Stop.

The left turn manoeuvre described above can be seen in Figure 4.1. The starting point is at the origin. While the vehicle is stationary, the steering angle is changed to align with the direction of the constant curvature circle arc, represented by the red arc. By keeping the steering angle fixed, the vehicle drives forward until the blue line is reached, and stops. The steering angle is then changed back to align with the heading of the vehicle. The vehicle drives forward for the length of the blue line and stops at the end.

The reason for stopping is to allow for the vehicle to change the angle of the steering wheels to be able to follow a constant curvature circle. If the steering controller was able to suddenly change the steer angle, the vehicle would be subjected to a sudden change in centripetal acceleration, causing unnecessary mechanical stresses and in a case of high speed driving, loss of traction. If these manoeuvres were attempted without stopping, it will cause the vehicle to deviate from the planned path, which means that the safety of the vehicle cannot be guaranteed any more. This method of planning and executing a path is impractical for most real world applications.

### 4.1.3 Continuously Variable Steer Angle Model

For this project, continuous driving was achieved by improving the steering model to include the executable change in steering angle while driving forward. The resulting path generated by the motion planner is a closer resemblance of the trajectory that the vehicle can execute. The manoeuvre library is therefore updated as follows:

- Left turn and straight.
- Right turn and straight.
- Straight.

The first manoeuvre implies that while the vehicle is driving, the steering angle increases at a constant rate from zero to a set steering angle and is kept at that angle for the duration of the required arc. The steering angle is then decreased to zero at a constant rate, while the vehicle continuous driving. After reaching a zero steering angle, the vehicle continuous driving straight for a required distance.

The second manoeuvre follows the same sequence and is a reflection of the first, where the steering angle is changed in the opposite direction and back to zero again.

The third manoeuvre is as it suggests, driving in a straight line, and can actually be realised from the first two by setting the change in steer angle and arc length to zero. It is however kept as a special manoeuvre to allow for the addition of forward acceleration or deceleration of the vehicle. This manoeuvre will typically be used at the start and end of the the vehicle's path, where starting to drive and stopping the vehicle is required.

Figure 4.1 illustrates the difference between the left turn manoeuvre of the simple Dubins car model and the improved manoeuvre that is implemented in this project.

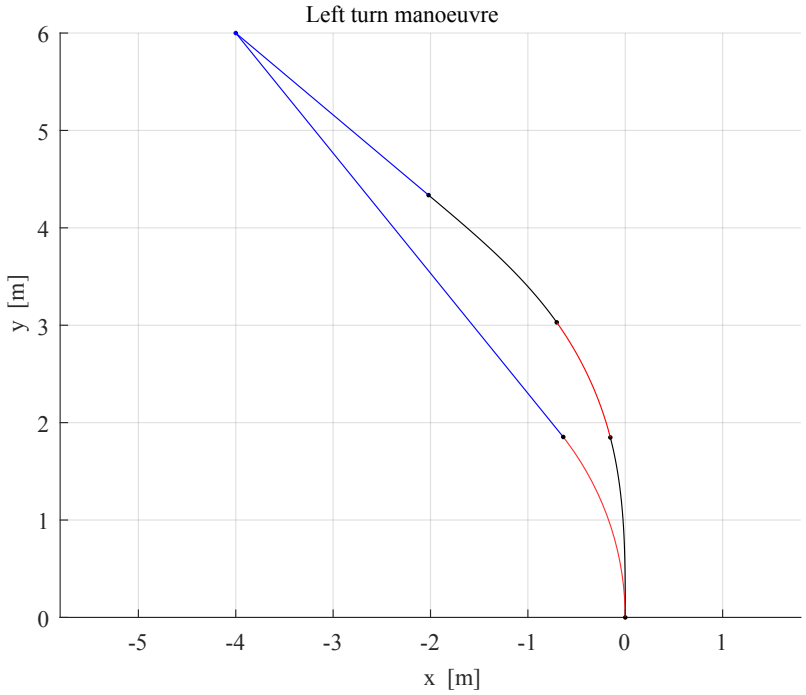
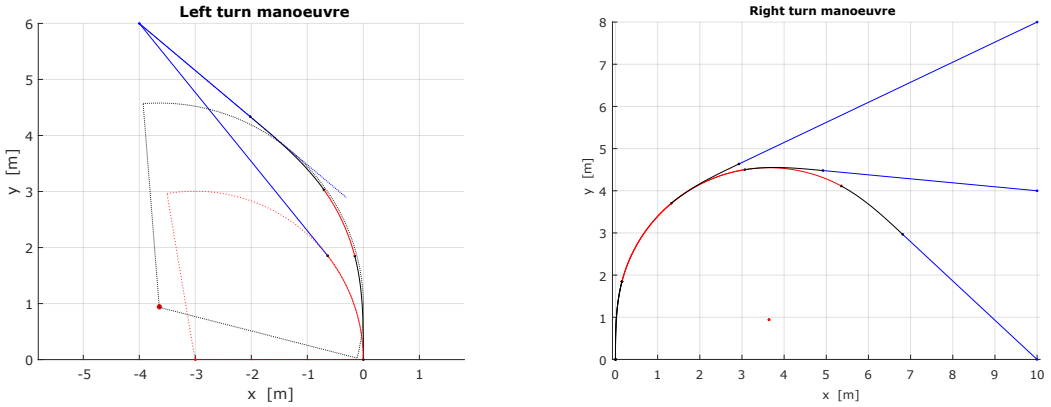



Figure 4.1: Comparison of left turn manoeuvres: Dubins car model (red and blue) and the continuously variable steer angle model (black, red and blue)

By using variable steering angles over time while moving forward, the resulting curves cannot be represented by only circles and straight lines any more. The transition between a zero steering angle and a constant steer angle, and vice versa, will have to be added. Figure 4.2(a) presents the effect of variable steering angle over time while driving and the resulting increase in turning radius where the straight line segment tangent connects. The red curve is a constant curvature circle segment. The black curves are the transition segments between the straight line (blue) and circle segments. Figure 4.2(b) shows three different right turn manoeuvres from the origin $(0, 0)$ to end points $(10, 0)$, $(10, 4)$ and $(10, 8)$, respectively.

A literature study on representing the transition between a linear and circular segment led to the field of highway and railway design [2]. The Euler spiral, also referred to as Cornu spirals or a clothoid, is introduced to be a two-dimensional curve of which the curvature is a linear function of the arclength of the curve. It is used to represent the transition between a curve and its tangent line.




(a) Increase in effective turning radius between the Dubins car model (red and blue) and the continuously variable steer angle model (black, red and blue)

(b) Right turn manoeuvres achieved by staying in the circular segment for longer using the continuously variable steer angle model

Figure 4.2: Additional manoeuvre representations: transition segments in black, circular segments in red and straight segments in blue

From the article presented by Connor and Krivodonova [11], the Euler spiral is described in a parametrised form as

$$\begin{pmatrix} x(\alpha) \\ y(\alpha) \end{pmatrix} = \begin{pmatrix} \bar{C}(\alpha) \\ \bar{S}(\alpha) \end{pmatrix}$$

where $\bar{C}(\alpha)$ and $\bar{S}(\alpha)$ are given by

$$\bar{C}(\alpha) = \frac{1}{\sqrt{2\pi}} \int_0^\alpha \frac{\cos(u)}{\sqrt{u}} \mathrm{d}u, \quad \bar{S}(\alpha) = \frac{1}{\sqrt{2\pi}} \int_0^\alpha \frac{\sin(u)}{\sqrt{u}} \mathrm{d}u$$

for positive values of $\alpha$. For negative values of $\alpha$, $\bar{C}(\alpha) = -\bar{C}(-\alpha)$ and $\bar{S}(\alpha) = -\bar{S}(-\alpha)$ is used. In these equations, $|\alpha|$ is the angle between the tangent and the x-axis. The expression for $\bar{C}(\alpha)$ and $\bar{S}(\alpha)$ were derived from the cosine and sine Fresnel integrals

$$C(t) = \int_0^t \cos\left(\frac{\pi u^2}{2}\right) \mathrm{d}u, \quad S(t) = \int_0^t \sin\left(\frac{\pi u^2}{2}\right) \mathrm{d}u$$

where $\bar{C}(\alpha)$ and $\bar{S}(\alpha)$ are related to $C(t)$ and $S(t)$ respectively by

$$\bar{C}\left(\frac{\pi t^2}{2}\right) = C(t), \quad \bar{S}\left(\frac{\pi t^2}{2}\right) = S(t).$$

Figure 4.3 plots the above equations for a visual representation of the Eular spirals in the negative and positive x-axis directions.

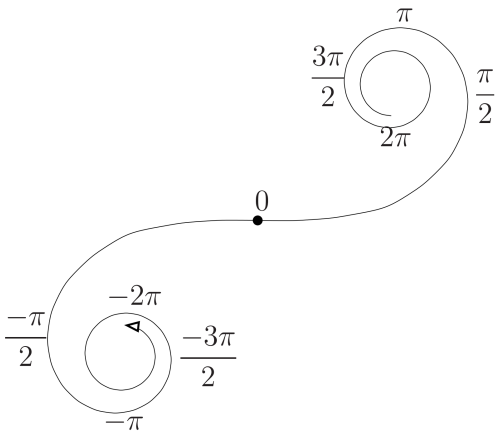



Figure 4.3: Eular spiral with $\alpha$ values labelled (source: Connor and Krivodonova [11])

For a constant vehicle velocity and rate of steering angle change, the transition between a zero steering angle and a constant steering angle $(0 \rightarrow \theta)$, can thus be created using the Euler spiral from the origin up to the point where $\alpha$ equals the steering wheel angle $\theta$. For the duration of driving with a constant steer angle $\theta$, the resulting length can be represented by an arclength of a circle. The length of transitioning between the constant steering angle back to a zero steering angle $(\theta \rightarrow 0)$ will be equal to the transition of $(0 \rightarrow \theta)$ which is already calculated. The remaining straight line segment can be calculated by the geometric difference between the point to be reached and the end of the $(\theta \rightarrow 0)$ transition segment.

Now that the transition segments have been defined, the algorithm that uses manoeuvres to solve a motion problem between two given points, can be described.

## 4.2 Local Planning Method

The algorithm that uses manoeuvres to solve a motion planning problem between two given points, $p1$ and $p2$, is referred to in this project as the local planning method (LPM). It should be noted that the LPM does not consider the environment nor performs any conflict detection. From the literature study done on path planning algorithms, an LPM takes a specific form and is used in the probabilistic roadmap method (PRM) as described

in Section 6.6.3. However, the $LPM$ algorithm developed for this project is a combination of the $STEER$ algorithm used in the rapidly-exploring random tree (RRT) path planner (Sections 6.6.1 and 6.6.2) and the $LPM$ algorithm as mentioned in the PRM path planner (Section 6.6.3). The difference between the local planning method (as used in the PRM) and steer method (as used in the RRT and RRT*) is summarized to be:

- $LPM$ considers end point heading and tries to connect $p1$ and $p2$ precisely
- $STEER$ ignores end point heading and only tries to come close to $p2$ (within predefined acceptable radius)

The planning method implemented in this project, Algorithm 1, contains properties of both of the above mentioned methods. The planning method tries to reach the end point precisely (as in the local planning method) but ignores the end heading (similar to the steer method). This ensures that the end goal can be reached precisely and assumes that the end point heading is not important or a requirement.

**Algorithm 1** LPM

**function** LocalPlanningMethod($p_1$,$p_2$)
  load the available **manoeuvres** from the $ManoeuvreLibrary$
  check if $p_2$ can be reached from $p_1$ with
  $Turn$ = CheckConnectivity($p_1, p_2, ManoeuvreLibrary$)
  **if** $Turn$ is not 'None' **then**
    **if** $Turn$ is 'Straight' **then**
      add a $StraightLine$ segment between $p_1$ and $p_2$
    **else**
      calculate the corresponding $SpiralSegments$ positions and orientations
      calculate the required $CircleArc$ between $SpiralSegments$
      add the remaining $StraightLine$ segment
    **end if**
    generate the $trajectory$ between $p_1$ and $p_2$ using $SpiralSegments$, $CircleArc$ and $StraightLine$
    **return** assigned $trajectory$ properties to **manoeuvre**
  **else**
    no manoeuvre available to connect $p1$ and $p2$
    **return** empty **manoeuvre**
  **end if**
**end function**

The sub-function, CheckConnectivity($p_1, p_2, ManoeuvreLibrary$), from Algorithm 1

above, takes the source point $p_1$, destination point $p_2$ and the updated manoeuvre library defined in Section 4.1.2 as inputs. From the manoeuvre library, the function can retrieve enough information from the manoeuvres (which encapsulates the vehicle and controller parameters) to determine if the destination point $p_2$ lies outside the two red shaded circles in Figure 4.4. If $p_2$ lies within the circles, it is too close and cannot be connected to $p_1$ with any of the manoeuvres. To do this check, the destination point $p_2$ is rotated and translated along with $p_1$ to have $p_1$ at the origin of the indicated x-y axis and the vehicle heading at $p_1$ rotated to align with the y-axis, shown by the blue line. If $p_2$ lies outside of the red shaded circles, it is reachable by both a left and a right turn manoeuvre. The function will then choose the turn manoeuvre that results in the shortest path length. If $p_2$ lies directly ahead of $p_1$, along the blue line, it is reachable by both left and right turn manoeuvres by turning 360 °in their respective directions, and the function will then rather choose a straight line manoeuvre.

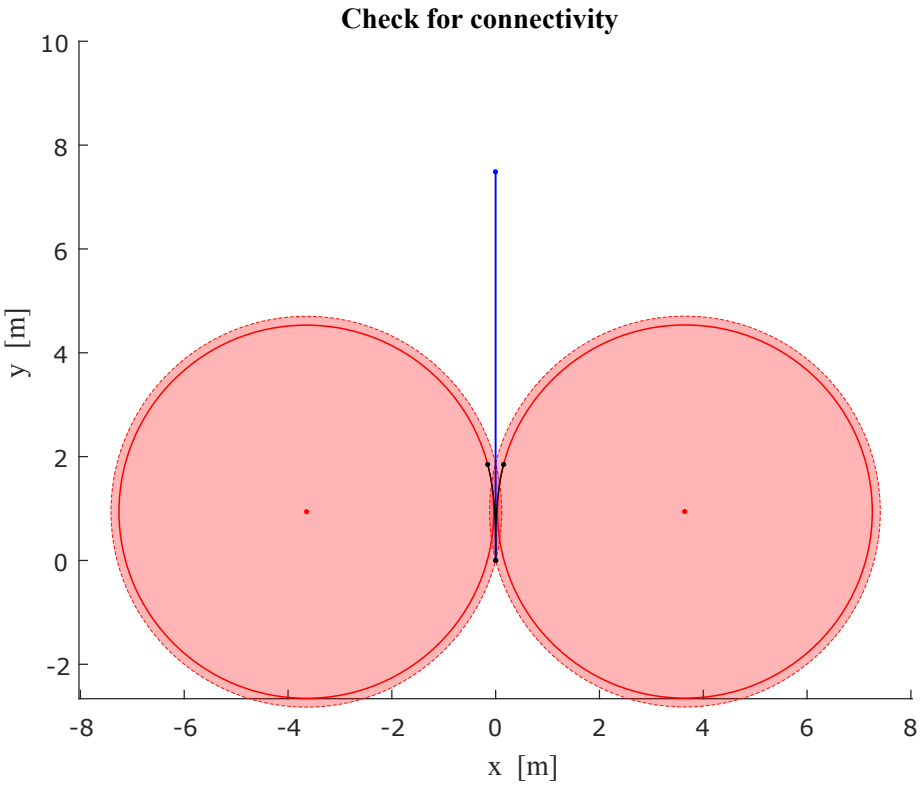



Figure 4.4: An illustration of the search area of CheckConnectivity()

The local planning method, discussed in this chapter, forms part of the implemented path planning algorithm of this project. The theory of path planning will follow in Chapter 6 and the implemented path planning algorithm is discussed in Chapter 7.

# Chapter 5

# Mapping and Conflict Detection

The path planning algorithm requires a searchable area to be provided within which it can do the path planning. While searching in the area, the path planner also requires some method of detecting if a path goes through an obstacle or if the distance between the path and obstacle is less than the minimum threshold specified.

The theory behind the Mapping Module and Conflict Detection Module is presented in this chapter. Mapping of the environment forms part of the autonomous navigation system; however, it is not the focus of this project. Some methods of representing the environment are presented, but practical set-up and implementation of the maps used will be provided in Chapter 7. Methods of detecting conflict or collisions are discussed, and follow after the mapping section.

## 5.1 Mapping

From the objectives of the project, listed in Section 1.3, it is necessary to represent the environment in the form of a map, and it must be searchable by the path planning algorithm. In a real-world application of an autonomous navigation system, full knowledge of the map is not necessarily available at the start of the planning procedure. The map will have to indicate this partial knowledge. The map must also have the ability to be updated as more information regarding the environment is received during the mission. This information usually comes from sensors on the vehicle and is relative to the position of the vehicle, not a global coordinate system of the map. Therefore the vehicle needs to localise itself within the map.

This section starts by discussing a method of combining the updating of the map and determining the relative position of the vehicle within the map. It then continues with a short survey of methods on representing a dense map. The section concludes with recent developments in representing a map as an adaptive occupancy grid and contains uncertainty in both the vehicle pose and the sensor measurements of the environment.

### 5.1.1 Simultaneous Mapping and Localization

Localizing the robot is typically performed by solving the simultaneous localization and mapping (SLAM) problem. The groundwork for SLAM was developed by Smith and Cheeseman in 1986 [50, 49] and then improved by a research group led by Durrant-Whyte in the early 1990's [40]. SLAM, and variants thereof, currently receive a great deal of research attention and consequently a lot of literature is available today covering this subject.

A SLAM system uses a sensor to "see" (such as stereo camera) and identifies landmarks in the environment. It then estimates the landmark locations and the autonomous vehicle's pose, relative to a fixed global axis system, by determining correspondences between landmarks over time. The SLAM system produces a sparse map due to the limitation of only identifying prominent landmarks. These maps do not include enough detail and are not sufficient for navigation, obstacle avoidance and path planning algorithms. Thus, it is necessary to fill the gaps between landmarks by building a dense map.

### 5.1.2 Dense Mapping of Static Environments

From the survey done by Thrun in 2002 [53], five dominant mapping algorithms are identified, namely Kalman filter approaches, expectation maximization (EM) algorithms, hybrid approaches of these two, object maps and occupancy grid maps. For a map to be useful it should be accessible to the path planning and conflict detection algorithms. Most modern mapping algorithms are probabilistic [53], which allows for the incorporation of uncertainty of the vehicle location and orientation as well as uncertainty about measurement information of the environment.

Kalman filter approaches: The Kalman filter approaches uses Bayes filters in an attempt to solve the SLAM problem and represents the posterior probabilities as Gaussian distributions. The maps that result from such approaches are generally represented as a point-cloud.

EM algorithms: Expectation maximisation algorithms work by maximising the posterior through a hill-climbing algorithm. These algorithms usually require multiple iterations through the data to reach convergence, and cannot be applied incrementally (while the autonomous vehicle moves in the environment). It should also be noted that this is an off-line algorithm.

Hybrid approaches: Hybrid approaches between the EM algorithm and the Kalman filter are known as incremental maximum likelihood algorithms [53]. These mapping algorithms

do not include uncertainty and builds a map of the environment incrementally as the sensor data is received.

Object maps: In object mapping algorithms the environment is represented in two dimensions by sets of line segments which implies that if the environment is too intricate to be represented by lines, they fail to generate a complete map [53]. This method has been extended to represent the geometry of the environment in three dimensions by using polyhedral sets. Some approaches build maps using a triangle mesh, but they cannot or have difficulties in representing uncertainty.

Occupancy grids: In this algorithm the region to be mapped is divided into a regular grid of cells and each is assigned an occupancy probability. The concept of occupancy grids was first introduced in 1985 by Moravec [41]. According to the survey done by Thrun [53], the occupancy grid algorithm provides distinct advantages over other static environment algorithms. The main shortcoming of the algorithm is that vehicle pose (location and orientation) uncertainty is not incorporated into the map, since the map is built by assuming that the most likely pose at a particular time step is correct.

### 5.1.3 Adaptive Occupancy Grids: 2D and 3D

As mentioned previously, occupancy grid mapping addresses the problem of generating a map from noisy and uncertain sensor measurement data. The initial algorithms were based on the assumption that the vehicle pose is known. This has since been extended to include uncertainty in the vehicle pose and is, among others, recently discussed by Joubert in 2012 [28]. Memory efficiency has been a problem of an occupancy grid since its development. The larger the mapping area, and the finer the grid resolution, the more memory it takes to represent an occupancy grid. When, for example, doing conflict detection on the map, the amount of data transfer and computational time becomes exponentially more. To solve the memory efficiency problem, adaptive occupancy grids were proposed which uses varying grid sizes. Payeur et al. 1997 implements this adaptive occupancy grid, also called an octree, to probabilistically model dynamic 3D environments [45].

### 5.1.4 Conclusion of mapping

This concludes the theory of representing maps of the environment. It is necessary to understand how maps are represented and searched to be able to develop a simplified map for the motion planning algorithm to test the implementation in this project. Updating the map and representing uncertainties in the map is beyond the scope of the project, and the chosen method of representing the map is geometrically and is discussed in Chapter 7.

## 5.2 Conflict Detection

As mentioned in the objectives of the project, Section 1.3, a collision detection method must be implemented to ensure that a feasible path from start to goal can be found. The function of the conflict detection module is to take the map representation as one input, and the suggested path segment from the path planner as the second input, to produce an output of whether conflict was detected or not. This is also confirmed by the literature study done on sampling based planning algorithms, Section 6.6. The collision, or conflict, detection module allows the path planning algorithm to be developed without knowledge of how the obstacle region is defined. Therefore, the type of conflict detection used is determined by the assumptions made regarding the environment and how it is defined. The conflict detection algorithm works by examining the trajectory points on a path between two nodes or states, and for each point the algorithm determines whether the point is outside of all the obstacles present in an environment and inside the map boundaries. If all these checks are *true*, the point is considered conflict free. Conflict detection methods can be grouped into two major types: deterministic and probabilistic.

### 5.2.1 Deterministic

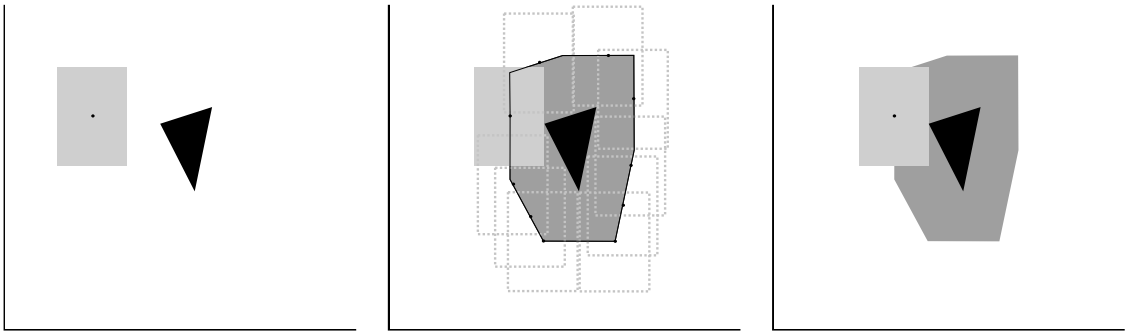
In the book *Planning Algorithms* by LaValle [38], a deterministic algorithm is distinguished from others by being an algorithm which, given a particular input, will always produce the same output without any uncertainty. A deterministic state is well defined, usually in two extremes, and contains no uncertainty: it is either *true* or *false*.

Deterministic conflict detection is based on the assumption that given a map of the environment, the open area is definitely open and the occupied regions are definitely occupied. For a map that does contain uncertainty, it can be made to work with this method by setting a threshold level on the allowed level of uncertainty. For example, if the uncertainty of the presence of an obstacle in the map, due to sensor measurement noise, is below the threshold, then there is no obstacle and the area is open.

Deterministic conflict detection is mainly used when a static environment is assumed, or if all information regarding a dynamic environment is readily available for any point in time: past, present and future.

A conflict region between a vehicle and obstacle can be described as any vehicle pose for which the vehicle and obstacle regions overlap. A popular method of determining a vehicle's allowable manoeuvring region in the environment is by using Minkowski addition [38]. The conflict region of an obstacle is expanded by adding the dimensions of the vehicle, and the vehicle dimensions are reduced to a unit. Figure 5.1 demonstrates the procedure that produces the conflict region around the obstacle. This conflict region will however be

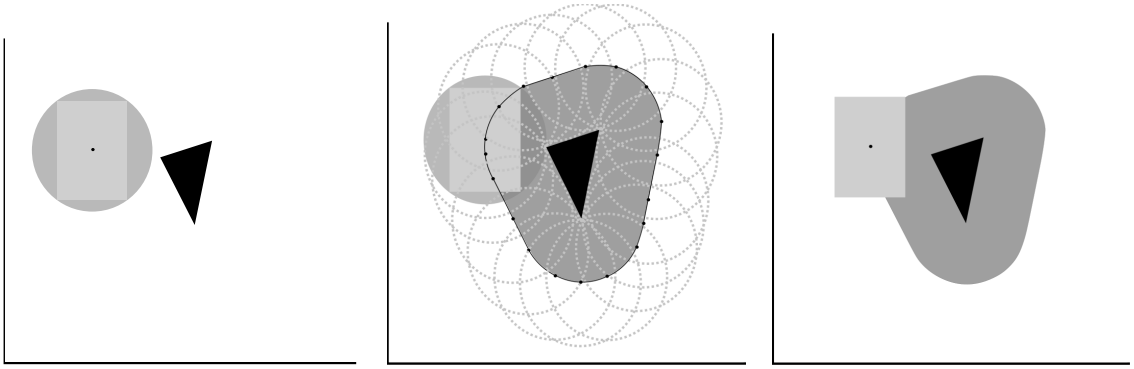
different for every possible orientation of the vehicle, as it can be seen when comparing figures 5.1(b), 5.4(a) and 5.4(b).



(a) Vehicle (grey) and obstacle (black) (b) Minkowski addition (c) Conflict area (dark grey)

Figure 5.1: Conflict area using Minkowski addition

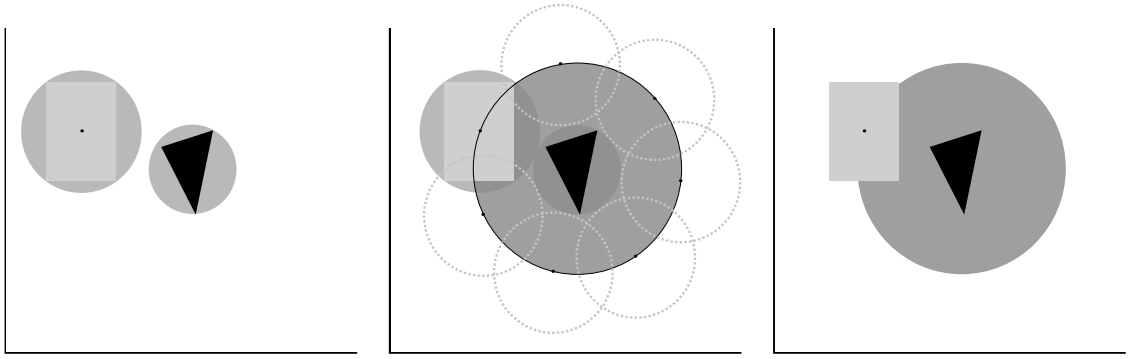
The computational time required for recalculating the obstacle region for every vehicle orientation before doing conflict detection can be removed by assuming a spherical form for the vehicle, or circular in the two-dimensional plane, as shown in Figure 5.2. The conflict region around all the known obstacles can now be calculated once and conflict detection can be done irrespective of how the vehicle will be approaching the obstacle.



(a) Vehicle (grey) and obstacle (black) (b) Minkowski addition (c) Conflict area (dark grey)

Figure 5.2: Conflict area for simplified vehicle pose

To simplify conflict detection even further, both the obstacle and the vehicle are defined to be circular in the two dimensional area map. Figure 5.3 illustrates the Minkowski addition for such a case. This simplifies the conflict detection problem to being a check for a point within a circle.



(a) Vehicle (grey) and obstacle (black)

(b) Minkowski addition

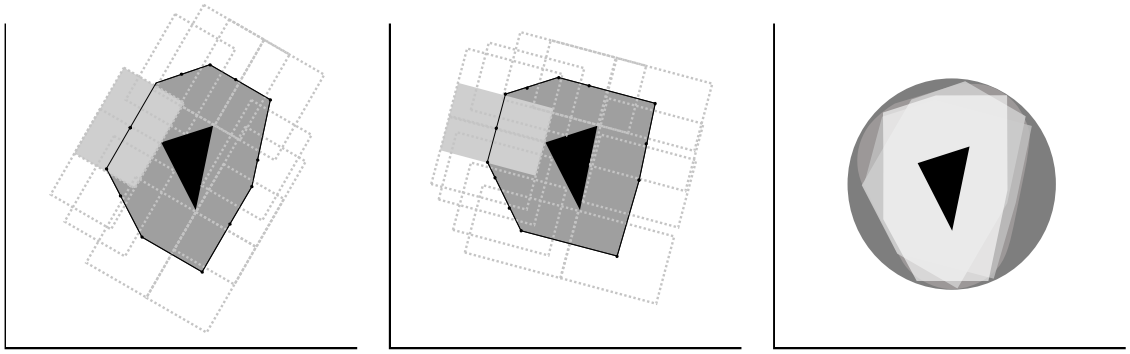
(c) Conflict area (dark grey)

Figure 5.3: Conflict area for simplified vehicle pose and obstacle uncertainty

To determine if a point with coordinates $(x, y)$ is within a circle with radius $R$ and centre coordinates $(x_i, y_i)$, the following will be true:

$$\Big((x - x_i)^2 + (y - y_i)^2\Big) \leq R^2$$

Lastly, Figure 5.4(c) presents a comparison of the different conflict areas for all the examples given in this section. It can be seen that the circle is an overestimation of the conflict area, and inherently provides a safety margin for uncertainties in both the vehicle and the obstacle pose.



(a) Vehicle orientation shifted by $\frac{\pi}{6}$

(b) Vehicle orientation shifted by $\frac{2\pi}{3}$

(c) A comparison of all of the conflict areas shown

Figure 5.4: Comparing conflict areas for different vehicle orientations

### 5.2.2 Probabilistic

To allow for uncertainties in the environment, the conflict detection algorithm needs to be probabilistic. To solve the probabilistic conflict detection problem, either a Monte Carlo method or a numeric and analytic method can be used.

Numerical and Analytic: Numerical and Analytic methods rely on the conflict detection problem to be discretized and defined in a closed form expression which can be solved by an approximation after a number of iterations. Paielli and Erzberger discusses the use of an analytic approach to conflict detection in 1997 and 1999 [42, 43]. More recently, Jones makes use of a numeric approach to conflict detection in 2003 and 2006 [26, 27].

Monte Carlo: Monte Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results [57]. This is usually done by running simulations many times over in order to calculate those same probabilities, just like actually playing and recording your results in a real casino situation: hence the name.

The Monte Carlo method is used in conflict detection to compute the probability of conflict through simulation of the conflict detection algorithm. It does not require a closed-form expression to be solved like the numerical an analytic methods mentioned previously. Due to the computational expensiveness of the Monte Carlo method, it is not implemented in real-time applications in it's original form. It is generally accepted to validate a conflict detection algorithm through Monte Carlo simulation. Yang and Kutcher published a paper in 2004 which uses the Monte Carlo method for conflict detection [57].

### 5.2.3 Conclusion of conflict detection

With the knowledge of different conflict detection methods, the chosen method to implement in this project is the deterministic conflict detection method. It is not required to present the outcome of the conflict detection with uncertainty because the inputs (the map of the environment and the path to be tested) are without uncertainty. Further implementation details is discussed in Chapter 7.

# Chapter 6

# Motion Planning Algorithms

Motion planning, or also known as the navigation problem, can be explained as a process which composes a solution to a task using discrete motions. In the case of autonomous navigation of a terrestrial vehicle, the motion planner will take the task of moving from one waypoint to the next as input, and produce the series of commands to the actuators required to execute the task. The motion planning algorithm will have to deal with the kinematic constraints of a terrestrial vehicle and uncertainty of partially mapped environments.

The book, *Planning Algorithms* [38], provides a broad coverage of the field of motion planning algorithms with the focus on vehicle motion planning, and is largely used as ground work for this chapter. It starts with an explanation of the concept of a configuration space and free space. It then continues to discuss how a planning algorithm is evaluated in terms of completeness. Lastly, it focuses on the different types of path finding algorithms.

## 6.1 Concepts of Motion Planning

The task of motion planning is to connect a start orientation and location with a goal orientation and location, by using a continuous motion and avoiding obstacles. The *workspace* describes the vehicle and obstacle geometry, and can be two- or three-dimensional. The *configuration space* describes the motion of the vehicle as a path in three dimensions or higher.

### Configuration space

A configuration describes the orientation and location of the vehicle, and the configuration space is the set of all possible configurations. In the case of a solid three dimensional shape that can translate and rotate, the workspace is then three-dimensional, but the configuration space is six-dimensional. Therefore, a configuration requires six parameters to be fully defined: $x$, $y$ and $z$ for translation, and Euler angles $\alpha$, $\beta$ and $\gamma$ for rotation. Fortunately, for a terrestrial vehicle, the degrees of freedom are restricted to the driving

surface, and this reduces the dimensions of the configuration space down to three, as mentioned by Visser [56].

### Obstacle region

The *free space* is the set of configurations that avoids collision with obstacles. The compliment of the free space in the configuration space is called the *obstacle region*. Obstacle regions can be explicitly or implicitly defined. The shape of the free space is generally not computed explicitly, but rather a collision detection test is done to determine whether a given configuration lies in the free space.

### Completeness

A motion planner is defined to be complete if the planner can find a solution or determine that there is definitely no solution, in a finite time [38]. The performance of a complete motion planner is measured by its computational complexity and time it takes to find a solution.

*Resolution completeness* is the property that the planner is guaranteed to find a path if the resolution of an underlying grid is fine enough. Most planners using grid based algorithms are resolution complete. The computational complexity of resolution complete planners is dependent on the number of points in the underlying grid.

*Probabilistic completeness* is the property that the longer the planner continues to search, the probability that the planner fails to find a path, if one exists, asymptotically approaches zero. Several planners using sampling-based algorithms are probabilistically complete. The performance of a probabilistically complete planner is measured by the rate of convergence.

*Incomplete* planners do not always produce a feasible path when one exists. However, they are sometimes used as real-time planners or reactive planners in dynamic environments where replanning is done regularly and execution time is limited [30].

## 6.2 Problem statement

This project concerns motion planning for a terrestrial vehicle, as described in Chapter 2, where the vehicle controllers are in place and map data can be sent to a conflict detection module. It is required that the vehicle can navigate safely from a starting position to an end goal, avoiding obstacles along the way. The motion planning algorithms should be able to take into account the manoeuvrability of the vehicle when planning a path, that is the kinematic and dynamic constraints.

In the sections to follow, three different groups of motion planning algorithms are presented. Each group will be evaluated against the problem statement, resulting in the motivation for the choice of path planning algorithm that is developed for this project.

## 6.3 Grid-Based Search

Motion planning algorithms can be classified according to how they search within a map. As the heading of this section implies, the environmental map is divided into a two- or three-dimensional grid, and each cell is labelled as occupied, empty or unknown. In Section 5.1, occupancy grid mapping is discussed in more detail. The algorithms based on this method of searching execute an incremental search on each and every block to determine if it is occupied or not. The most common examples of these search algorithms are the A* and D* algorithms and variants thereof. Grid based search is typically a very computational expensive search algorithm, but is proven to be resolution complete, as shown by the survey of algorithms done by Ferguson et al. in 2005 [14].

### 6.3.1 A*

The A* algorithm, pronounced A-star, was first described in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute [23], and is an extension of Edsger Dijkstra's 1959 algorithm. This algorithm is rather old and already outperformed by more recent algorithms [12], but is still in use and therefore is included into the literature review done for this project.

The A* algorithm, pseudocode given in Algorithm 2, searches the finite configuration space to find a path in the form of $S$, which is a set of states $s$, from the initial state $s_{start} \in S$ to the goal state $s_{goal} \in S$.

The A* algorithm, pseudocode given in Algorithm 2, plans a path in $S$, which is a set of states $s$ in the finite configuration space, from the initial state $s_{start} \in S$ to the goal state $s_{goal} \in S$. From the initial state to each state $s$, the path cost $g(s)$ is estimated and stored. In line 3, the initial estimated path cost is $g(s) = \infty$ for all states $s \in S$. In line 5, the algorithm starts by updating the start state's path cost to be zero, and then places this state onto a priority queue, namely, the **OPEN** list. This queue is then sorted according to the the sum of an element $s$'s current path cost from the start $g(s)$ and a heuristic estimate of its path cost to the goal, $h(s, s_{goal})$.

Always located at the front of the priority queue is the state with the minimum value of the total cost $g(s) + h(s, s_{goal})$. The estimated path cost to the goal $h(s, s_{goal})$ is used to guide the search and typically underestimates the cost of the optimal path from $s$ to $s_{goal}$.

In line 12, the algorithm then pops the state $s$ at the front of the queue and updates the cost of all states reachable from this state through a direct edge. In lines 13 to 16, if the cost $g(s)$ of the state $s$ with the edge cost $c(s, s')$ added, between state $s$ and a neighbouring state $s'$, is less than the current cost of state $s'$, then the cost of $s'$ is set to this lower value.

If the cost of a neighbouring state $s'$ changes, it is placed on the **OPEN** list. The algorithm continues adding states, sorting the queue and popping states off until it pops off the goal state. When this occurs, if the heuristic is *admissible*, as described in [23] to mean that it is guaranteed to not overestimate the path cost from any state to the goal, then the path cost of $s_{goal}$ is guaranteed to be optimal. The complete algorithm is summarised in Algorithm 2, adapted from Ferguson et al. [14]. It should be noted that the A* algorithm can also be changed to search backwards from the goal state to the start state.

**Algorithm 2** A*() - forward version

```
1:  function MAIN()
2:      for all s ∈ S do
3:          g(s) = ∞
4:      end for
5:      g(s_start) = 0
6:      OPEN = ∅
7:      insert s_start into OPEN with value (g(s_start) + h(s_start, s_goal))
8:      COMPUTESHORTESTPATH()
9:  end function

10: function COMPUTESHORTESTPATH()
11:     while (argmin_{s∈OPEN}(g(s) + h(s, s_goal)) ≠ s_goal) do
12:         remove state s from the front of OPEN
13:         for all s' ∈ Successors(s) do
14:             if (g(s') > g(s) + c(s, s')) then
15:                 g(s') = g(s) + c(s, s')
16:                 insert s' into OPEN with value (g(s') + h(s', s_goal))
17:             end if
18:         end for
19:     end while
20: end function
```

The above approach works well for planning an initial path through a known grid or workspace, with the assumption that the environment is static or that information regarding the environment does not change. However, when working in real world

scenarios, an autonomous vehicle typically does not have perfect information regarding its surroundings. So, every time additional or new information is received from an environmental sensor which changes or updates the map, a new path must be re-planned from scratch. This procedure is a waste of computational resources. The following variants of the D* algorithm handles the replanning by distinguishing when new information does not affect the initial path or only part of the path, in the former case not doing any replanning and for the latter only re-plans around the influenced part of the path.

### 6.3.2 D*, Focused D* and D* Lite

The D* algorithm, pronounced D-star, was developed by Anthony Stentz in 1994 [51] and is an informed incremental search algorithm. The name D* comes from the term dynamic A*, because the algorithm behaves like A* except that the path costs can change as the algorithm runs due to updated information from sensors. Same as A*, the D* algorithm also makes use of the $OPEN$ list priority queue to order nodes according to their path cost. However, the D* algorithm also keeps track when a node's cost changes, and then only re-evaluates around that node.

The focused D* algorithm resulted from further development of D* by Anthony Stentz in 1995 [52], and is an informed incremental heuristic search algorithm that combines A* and D*. The focused D* algorithm uses a heuristic to focus the propagation of evaluating the nodes which had an increase or decrease in their path costs due to updated information from environmental sensors.

The D* Lite is an incremental heuristic search algorithm developed by Sven Koenig and Maxim Likhachev in 2002 [33]. It combines ideas from A* and Ramalingam and Reps's dynamic graph algorithm [47]. D* Lite is not based on D* or focused D*, but rather implements the same behaviour. Performance-wise, it is better than focused D* and simpler to understand with fewer lines of code to implement [14]. It is for this reason that only the D* Lite algorithm is explained in further detail.

The D* Lite algorithm, pseudocode given in Algorithms 3 and 4, starts by calculating an optimal path from the initial state to the goal state in exactly the same way as the backwards A* algorithm. When updated sensor information is received and the map is changed, the path costs (of states whose paths to the goal are directly affected) are updated. These updated states are then placed on the $OPEN$ list priority queue. By doing this, the effects of the changes are propagated to the rest of the states as the queue is handled. This ensures that only the portion of the state space, that is affected by the new map information, is processed when changes occur.

Also, D* Lite uses a heuristic to further focus the states processed to only those whose change in path cost could have a bearing on the path cost of the initial state. Similar

to A*, D* Lite plans a path in $S$, which is a set of states $s$ in the finite configuration space, from the initial state $s_{start} \in S$ to the goal state $s_{goal} \in S$. It also stores a one-step lookahead cost [33], $rhs(s)$, which satisfies:

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{goal} \\ \min_{s' \in Successor(s)}(c(s, s') + g(s')) & \text{otherwise} \end{cases},$$

where $Successor(s) \in S$ denotes the set of successors of $s$ and the edge cost from $s$ to neighbour $s'$ is given by $c(s, s')$. From [33], a state is called consistent if and only if $g(s) = rhs(s)$, otherwise it is either over-consistent if $g(s) > rhs(s)$ or under-consistent if $g(s) < rhs(s)$. Again similar to A*, D* Lite uses a heuristic to focus its search and order the cost updates in a priority queue. The estimated cost of moving from state $s$ to $s'$ is given by the heuristic $h(s, s')$. It has to be admissible and backwards consistent: $h(s, s') \leq c^*(s, s')$ and $h(s, s'') \leq h(s, s') + c^*(s', s'')$ for all states $s, s', s'' \in S$, where the cost associated with the least-cost path from $s$ to $s'$ is given by $c^*(s, s')$. The $OPEN$ list priority queue always holds the inconsistent states, and these states are exactly the states that need to be updated and made consistent. The priority, or *key* value, of a state $s$ in the queue is:

$$\text{KEY}(s) = \begin{bmatrix} k_1(s) & k_2(s) \end{bmatrix},$$

where

$$k_1(s) = \min(g(s), rhs(s)) + h(s_{start}, s)$$
$$k_2(s) = \min(g(s), rhs(s)).$$

A lexicographic ordering is used on the priorities, which means that $\text{KEY}(s) \leq \text{KEY}(s')$, if and only if $k_1(s) < k_1(s')$ or both $k_1(s) = k_1(s')$ and $k_2(s) \leq k_2(s')$.

D* Lite expands states from the queue, sorted from with lowest to highest priority, updating their $g(s)$ values and their predecessors' $rhs(s)$ values, until all the states in the queue have a priority higher than the start state. Thus, during its calculation of an initial path, it performs in exactly the same way as a backwards A* search. For interest, the main reason why the D* Lite algorithm searches backwards from the goal to the start is to allow for the possibility that the start state may change over time. Therefore, the autonomous vehicle's current position always represents the start state, and only the heuristic value associated with each inconsistent state needs to be updated when the vehicle moves. When the edge costs $c(s, s')$ change, D* Lite updates the $rhs(s)$ values of each state directly affected by the changed edge costs and places the states onto the $OPEN$ list priority queue that have been made inconsistent. As before, it then expands the states

on the queue in order of increasing priority until all the states in the queue have a priority higher than the start state.

**Algorithm 3** D* Lite() - Part 1

```
1:  function MAIN()
2:      g(s_start) = ∞
3:      g(s_goal) = ∞
4:      rhs(s_start) = ∞
5:      rhs(s_goal) = 0
6:      OPEN = ∅
7:      insert s_goal into OPEN with key(s_goal)
8:      loop
9:          ComputeShortestPath()
10:         Wait for changes in edge costs
11:         for all directed edges (u, v) with changed edge costs do
12:             Update the edge cost c(u, v)
13:             UpdateState(u)
14:         end for
15:     end loop
16: end function
```

**Algorithm 4** D* Lite() - Part 2

17: **function** KEY(s)
18: $k_1(s) = min(g(s), rhs(s) + h(s_{start}, s))$
19: $k_2(s) = min(g(s), rhs(s))$
20: **return** $\begin{bmatrix} k_1(s) & k_2(s) \end{bmatrix}$
21: **end function**

22: **function** COMPUTESHORTESTPATH()
23: **while** ($min_{s \in \textbf{OPEN}}$(KEY($s$)) $<$ KEY($s_{start}$) **or** $rhs(s_{start}) \neq g(s_{start})$) **do**
24: remove state $s$ with the minimum key from **OPEN**
25: **if** ($g(s) > rhs(s)$) **then**
26: $g(s) = rhs(s)$
27: **for all** $s' \in Predecessor(s)$ **do**
28: UpdateState($s'$)
29: **end for**
30: **else**
31: $g(s) = \infty$
32: **for all** $s' \in Predecessor(s) \cup \{s\}$ **do**
33: UpdateState($s'$)
34: **end for**
35: **end if**
36: **end while**
37: **end function**

38: **function** UPDATESTATE()
39: **if** $s$ was not visited before **then**
40: $g(s) = \infty$
41: **end if**
42: **if** ($s \neq s_{goal}$) **then**
43: $rhs(s) = min_{s' \in Successor(s)}(c(s, s') + g(s'))$
44: **end if**
45: **if** ($s \in$ **OPEN**) **then**
46: remove $s$ from **OPEN**
47: **end if**
48: **if** ($g(s) \neq rhs(s)$) **then**
49: insert $s$ into **OPEN** with key($s$)
50: **end if**
51: **end function**

Algorithms 3 and 4 are adapted from the survey of Ferguson et al. [14] and show a basic version of the D* Lite algorithm which uses a fixed start state. A more efficient implementation of the algorithm can be found in the paper by Koenig and Likhachev [33].

### 6.3.3 Conclusion of grid based search

This group of algorithms are not considered. The first reason is that in Section 5.1 it is mentioned that the map will be a simplified geometric representation of the environment, with implementation detailed in Chapter 7. The grid-based search algorithms are restricted to maps that are divided into a grid. Another reason is that the algorithms discussed in this section do not include the vehicle's kinematic and dynamic constraints.

In the following section the potential field methods are discussed.

## 6.4 Potential Fields

Potential field methods originated from the concept of assigning a potential function to the free space, and simulating the vehicle as a particle reacting to forces due to the potential field [38]. The goal point has the lowest potential, and attracts the vehicle, while obstacles repel the vehicle. Potential field methods were adapted by Rimon and Koditschek in 1988 [48] to be applied to the navigation problem, but resulted in an incomplete path planner. Since then, two classes of potential fields have been defined that satisfy the properties of a navigation function. The first is based on harmonic functions as proposed by Connolly et al. [10] and the second based on solving the optimal distance-to-go function by Guillemin and Pollack [22]. Both of these methods are subject to the local minima problem and can become trapped. The randomized potential-field planner is the result of further developments done.

### 6.4.1 Randomized Potential-field Planner

The randomized potential-field planner is designed to help a potential field search algorithms escape from local minima that are found in the original potential field method.

One of the methods of randomizing the potential field planner is to let the planner take random walks whenever it gets stuck in a local minima. The number of times that a random walk can be taken is limited by $K$. To determine if the planner is stuck in a local minima, the number of iterations that fail to optimise the cost function $g$ are limited by $L$. Another parameter, $M$, defines the maximum length of the random walk that can be taken.

An example of the randomized potential-field planner is adapted from [38] and summarised in Algorithms 5 and 6 for reference.

**Algorithm 5** Randomized Potential-field Planner - Part 1

```
1:  function MAIN()
2:      K = x, L = y, M = z
3:      i = 1
4:      Start potential function for g(s_start → s_goal)
5:      Enter state BestFirst

6:      state BestFirst
7:      j = 0
8:      for all j < L do
9:          create v_n ∈ V in neighbourhood of v_latest
10:         if g(v_n) < g(v_latest) then
11:             v_latest ← v_n and v_latest to path
12:             j = 0
13:         else
14:             j++
15:         end if
16:     end for
17:     if i < K then
18:         Go to state RandomWalk
19:     end if
20:     if i == K then
21:         Go to state BackTrack
22:     end if
```

**Algorithm 6** Randomized Potential-field Planner - Part 2

**state** RandomWalk
$i++$
$k = 0$
**for all** $k < M$ **do**
    execute random walk from $path(v_{latest})$
    **if** $g(v_n) < g(v_{latest})$ **then**
        Go to **state** BestFirst
    **else**
        $k++$
    **end if**
**end for**
go to **state** BestFirst

**state** BackTrack
select $random(v \in V)$ generated during **state** RandomWalk
Reset $i = 1$ and go to **state** BestFirst
**end function**

### 6.4.2 Conclusion of potential fields

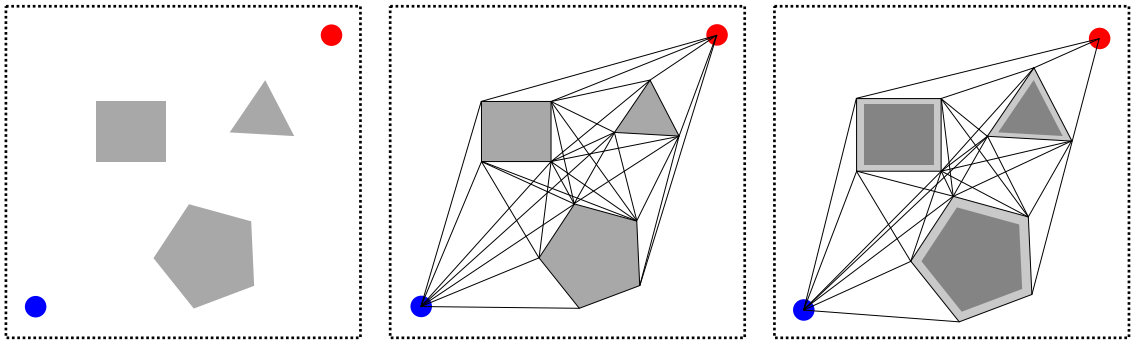
Barraquand and Latombe, among others, have done extensive development on randomized potential field planners from 1990 to 1995 [3, 4, 35] and Ge and Cui more recently in 2002 [18]. By adding random components to subsequent steps or changing the values of obstacle potentials at random, the search can be randomized. Completeness of these algorithms has been difficult to prove, although probabilistic completeness can be achieved. The randomized potential field approach can escape high-dimensional local minima and provides solutions to many challenging problems, but requires multiple parameter tuning for specific applications [38]. Due to this reason, newer and more recent methods are preferred to the potential field method.

## 6.5 Geometric Methods

The following search algorithms are geometrically based, where the environment map is drawn in the form of a graph. This approach is also referred to as the roadmap method, and requires an exact map of the environment to work. Since the initial project definition stated that the environment is only partially known, this group of algorithms were not considered and therefore will only be described briefly. The conclusion of this section includes an evaluation of the planning methods against the problem statement.

### 6.5.1 Visibility Graph

The visibility graph produces a complete and optimal solution to the navigation problem in a two dimensional configuration space [21]. It works by assuming that the shortest path grazes obstacles at their vertices. For a search space given in Figure 6.1(a), a roadmap of lines is built connecting each vertex with all vertices visible from its position (Figure 6.1(b)), hence the name. To prevent collisions with an obstacle which has uncertainty in its position or orientation, the obstacle region can be expanded with a safety buffer region (Figure 6.1(c)).

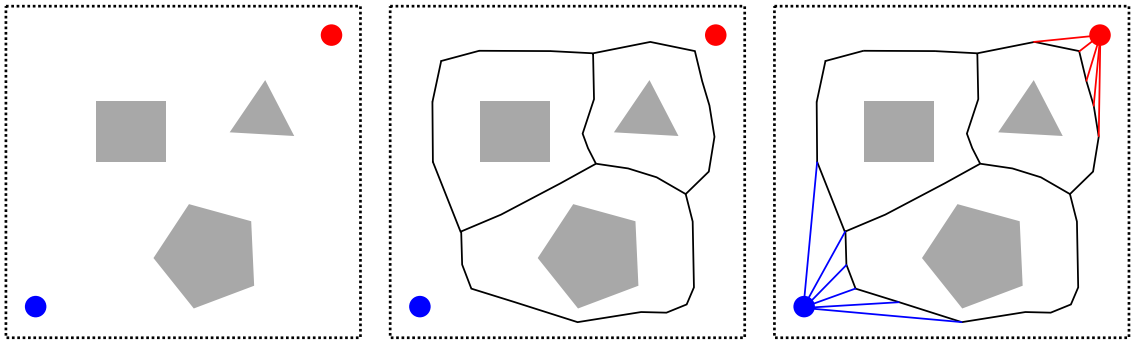


(a) Search space (b) Obstacles without a safety buffer (c) Obstacles with a safety buffer

Figure 6.1: Constructing a visibility graph for a given map

### 6.5.2 Voronoi Diagram

All roadmap methods pose the problem of controlling a vehicle accurately enough to follow a minimum distance path without collision with obstacles, not to mention obstacles with uncertainty in their position. The Voronoi roadmap method overcomes this problem by building a skeleton (Figure 6.2(b)) that is maximally distant from the obstacles (Figure 6.2(a)), and finds the minimum distance path that follows this skeleton (Figure 6.2(c)). The solution is no longer optimal, but is complete. The survey done by Aurenhammer in 1991 [1] covers algorithms that compute Voronoi diagrams. The Voronoi diagram method have also been expanded to handle multiple dimensions by Choset and Burdick in 2000 [7] and later updated to allow incremental construction [8]. More recently, Howlett et al. 2004 [24] discusses the use of Voronoi roadmap methods for unmanned rotary-wing aircraft.

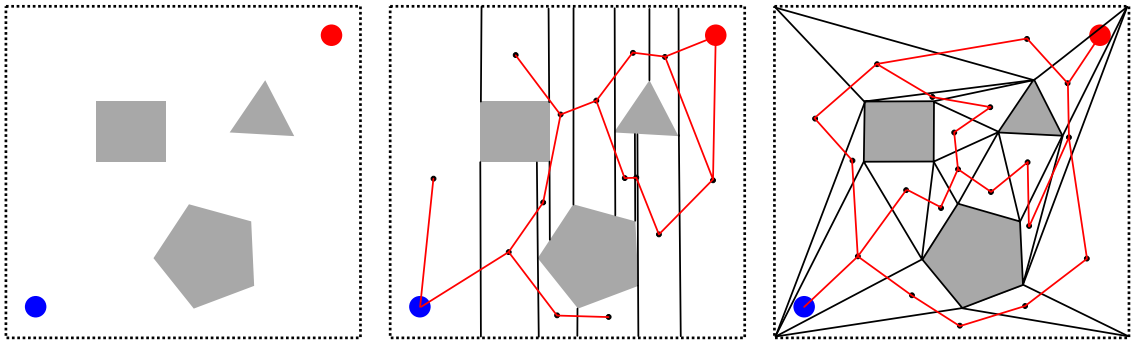


(a) Search space (b) Voronoi diagram (c) Voronoi diagram with start and end points connected

Figure 6.2: Constructing a voronoi diagram for a given map

### 6.5.3 Cell Decomposition

Cell decomposition is not a path planning algorithm, but rather a method of decomposing the free configuration space into smaller convex polygons, which are then connected by a graph and searched using a graph search algorithm [21]. Figure 6.3 illustrates two examples of using exact cell decomposition on the search space. Vertical decomposition, Figure 6.3(b), is done by dividing the search space into trapezoidal areas. Vertical lines are extended from each of the obstacle vertices, stopping at the first obstacle or boundary it reaches. A roadmap is then constructed by joining the centre points of neighbouring trapezoids. Triangulation, Figure 6.3(c), constructs triangular areas between obstacle vertices and boundary corners, from which a roadmap can be formed by connecting the midpoints of neighbouring triangles.



(a) Search space (b) Vertical decomposition (c) Triangulation

Figure 6.3: Examples of cell decomposition for a given map

### 6.5.4 Conclusion of geometric sampling methods

These algorithms use a similar representation of the map used for the project, but the main downfall of these planning methods is that they do not take into account the manoeuvrability of the vehicle, meaning that the generated path does not include the kinematic and dynamic constraints of the vehicle. Because the vehicle will be unable to follow the path, the planner can no longer ensure that the vehicle can navigate safely from the start to the end goal. These planning methods therefore fail the requirements of a path planner for this project.

## 6.6 Random Sampling

Sample-based motion planning algorithms eliminate the need to explicitly construct the obstacle region, and instead conduct a search that probes the configuration space with a sampling scheme. This probing is enabled by a collision detection module, which is independent from the motion planning algorithm. This results in the ability to develop planning algorithms which are independent of the particular geometric models [38].

### 6.6.1 Rapidly-exploring Random Tree

The rapidly-exploring random tree was first introduced by Steven LaValle in 1998 [36] and then in the following year it was expanded to include kinematic constraints [39]. It was then already capable of handling nonholonomic, dynamic constraints in search spaces with high degrees of freedom. LaValle compares RRT with dynamic programming in his publication [37]. The RRT algorithm expands by randomly selecting a point, or node, from the configuration space and tries to connect it to the nearest segment on the existing tree via a path segment, which is achieved by applying control inputs to the system or using manoeuvres. This path segment is tested for conflict and if none is found, the randomly selected point with connecting path segment is added to the tree. Otherwise, if conflict is predicted, the path segment stops at the point of conflict and the randomly selected point is discarded. The RRT is probabilistically complete and does not attempt to find the optimal path, but rather a feasible path. In 2010, Karaman and Frazzoli [31] provided a mathematical proof that the cost of the best path in the RRT converges almost certainly to a non-optimal value.

The following section details the improvements on the RRT algorithm.

### 6.6.2 Rapidly-exploring Random Graph with Tree characteristics

The rapidly-exploring random graph (RRG) is a new algorithm which was developed in 2010 by Karaman and Frazzoli [31], and is proven that the cost of the best path in the RRG converges almost certainly to an optimal value. The RRG was then adapted into a tree version by the authors, referred to as the RRT*, which has the optimality of the RRG and the tree structure of an RRT. Karman and Frazzoli also proved that the computational complexity of the RRT* is essentially the same as that of the RRT.

The RRT* algorithm works by "rewiring" the tree structure as it discovers new lower-cost paths reaching the nodes that are already in the tree. The RRT* algorithm differs from the RRT and the RRG algorithms only in the way that it handles the expanding procedure to grow the tree. Similar to RRT and the RRG, the RRT* algorithm first extends the nearest neighbour towards the random sample. However, it connects the new path segment to a segment in the tree which incurs the minimum accumulated cost up until the new path segment. Algorithm 7 shows the implementation of RRT*, which was adapted from the work of Karman and Frazzoli [31] and the project done by C.Beyers [5].

**Algorithm 7** RRT* - Part 1

```
1: function PathPlanner(s_start, s_goal, S)
2:     initialise tree_RRT*
3:     s' = s_start
4:     while algorithm should keep searching do
5:         use Steer(s', s_goal) to try and connect s' to s_goal
6:         test for conflict between s' and s_goal
7:         if the path between s' and the s_goal is conflict free then
8:             if the cost to the goal is improved then
9:                 save the new path to the goal
10:            end if
11:            if algorithm should stop as soon as a path is found then
12:                return path to the goal
13:            end if
14:        else
15:            if number of iterations exceeds limit then
16:                return path to the goal
17:            end if
18:            if number of points exceeds limit then
19:                return path to the goal
20:            end if
21:            if execution time exceeds limit then
22:                return path to the goal
23:            end if
24:        end if
25:        generate new s_random_sample
26:        use ExtendRRT* to insert s_random_sample into tree_RRT*
27:        s' = returned by ExtendRRT*
28:    end while
29: end function
```

**Algorithm 8** RRT* - Part 2

**function** STEER($s$,$s'$)
    determine how the new point $s'$ can be reached from the nearest neighbour $s$
    use the manoeuvre library to get as close as possible to the new point $s'$
    move $s'$ to coincide with the end of the manoeuvre
**end function**

**function** EXTENDRRT*($tree_{RRT^*}$,$s'$,$s_{random\_sample}$,$s_{goal}$,$S$)
    use STEER($s'$, $s_{random\_sample}$) to try and connect $s'$ to $s_{random\_sample}$
    test for conflict on the path between $s'$ and $s_{random\_sample}$
    **if** the path is conflict free **then**
        **for** each $s$ in $tree_{RRT^*}$ **do**
            $distance$ = distance between $s$ and $s_{random\_sample}$
            **if** $distance < \eta$ **then**
                $temp\_s$ = copy of $s_{random\_sample}$
                use STEER($s, temp\_s$) to try and connect the points
                test for conflict on path between $s$ and $temp\_s$
                **if** another nearest neighbour $s$ is found with lower cost than the cost of $s'$ to $temp\_s$ **then**
                    cost of $s_{random\_sample}$ = cost of $temp\_s$
                    path of $s_{random\_sample}$ = path of $temp\_s$
                **end if**
            **end if**
            add $s_{random\_sample}$ to $tree_{RRT^*}$
            $s' = s_{random\_sample}$
        **end for**
    **end if**
**end function**

Using the same notation as for the A* and D* Lite algorithms, the RRT* algorithm plans a path in $S$, which is a set of states, milestones or nodes, $s$ in the configuration space, from the initial state $s_{start} \in S$ to the goal state $s_{goal} \in S$. All nodes $s$ which can be connected and are collision free are added to the tree $tree_{RRT^*}$. The PATHPLANNER($s_{start}, s_{goal}$) function iterates until a path is found or until the number of iterations, points or execution time have been exceeded. The function STEER($s, s'$) generates a path segment from the current point $s$ towards the new point $s'$ using a library of manoeuvres which takes into account the kinematic constraints of the vehicle. The path segment is not required to reach the new point exactly, but only to come within close proximity. The EXTENDRRT*($tree_{RRT^*}$,$s'$,$s_{random\_sample}$,$s_{goal}$,$S$) function tries to connect the randomly

sampled point $s_{random\_sample}$ to the nearest segment $s'$ on the tree of points $tree_{RRT^*}$. If the random sample is close enough to be added, within the radius of length $\eta$, it also checks for a lower cost path in the area.

The following sampling-based algorithm is similar to the RRT* algorithm, and is therefore noteworthy to discuss.

### 6.6.3 Probabilistic Roadmap Method

The probabilistic roadmap method was first introduced by Kavraki et al. in 1996 [32]. A survey of PRM methods were done by Geraerts and Overmars in 2004 [20]. The PRM algorithm and its variants are multiple-query methods that first construct a graph (the roadmap), which represents a large portion of the free space, and then answers queries by computing the shortest path that connects the initial state with the goal state through the graph. The PRM differs from the previously mentioned roadmap methods (visibility graph and voronoi diagrams) in that it selects random points in the configuration space, irrespective of obstacles, and then tries to connect points with paths that are conflict free. Also differently from RRT*, the PRM algorithm tries to connect a random sample exactly to a known reachable point, and continues to attempt the connection to all reachable points until it finds a path to the random sample. The algorithm makes use of a local planning method (LPM), which implements a library of manoeuvres, to do this path generation from the random sample to the known reachable points. When a feasible path to the goal is found, the algorithm continues to generate random samples and tries to find a path with a lower cost function until a time limit is reached.

The PRM algorithm is shown in Algorithm 9 and is adapted from [5]. It takes the same form as the RRT* algorithm, with the exception of the LPM function which connects two points exactly instead of the STEER function which only steers from one point towards the next point without reaching it exactly. Also, the EXTENDPRM function only searches the first $n$ number of sorted points in the $tree_{PRM}$, whereas the EXTENDRRT* function is limited by the distance $\eta$ it can extend from the $tree_{RRT^*}$.

**Algorithm 9** PRM

```
function PathPlanner(s_start,s_goal,S)
    initialise tree_PRM, s' = s_start
    while algorithm should keep searching do
        use the LocalPlanningMethod to try and connect s' to s_goal
        test for conflict between s' and s_goal
        if the path between s' and the s_goal is conflict free then
            if the cost to the goal is improved then
                save the new path to the goal
            end if
            if algorithm stop condition is satisfied then
                return path to the goal
            end if
        else
            generate new s_random_sample
            use ExtendPRM to insert s_random_sample into tree_PRM
            s' = returned by ExtendPRM
        end if
    end while
end function

function LocalPlanningMethod(s,s')
    for each manoeuvre in the manoeuvre library do
        calculate the path which minimises the cost function between s and s'
        store this path in the end point s'
    end for
end function

function ExtendPRM(tree_PRM,s',s_random_sample,s_goal,S)
    sort the list of reachable points in the tree_PRM by distance to s_random_sample
    for the first n points in the sorted list of the tree_PRM do
        let s be the next point in the list of points of tree_PRM
        connect s_random_sample to s using LocalPlanningMethod
        test the path between s_random_sample and s for conflict
        if conflict free then
            add s_random_sample to tree_PRM
            nearest neighbour to s_random_sample with no conflict is s
            latest point s' = s_random_sample
            return
        end if
    end for
end function
```

### 6.6.4 Conclusion of random sampling methods

From the different groups of motion planning algorithms, the sampling-based motion planners fit the requirements of the problem statement the best. Sample-based motion planning algorithms do not require knowledge of the map and obstacle representation to be developed. These algorithms achieve this by requiring a supplementary function to process the conflict detection. The path that it builds by successfully connecting points, can also be done by a supplementary function to determine the form of the path. The supplementary function that builds the path can include specified constraints, for example the kinematic and dynamic constraints of a vehicle, to guarantee that the path it produces can be followed by the vehicle.

By looking at the requirements of the problem statement, the sample-based motion planning algorithms can be applied to a terrestrial vehicle to navigate it safely (by generating a path that the vehicle can follow) and avoiding obstacles (with the supplementary conflict detection function).

The modularity of the sample-based algorithms allow them to fit in the AutoNav framework, discussed in Section 1.2. This allows the focus of the project to be applied to developing a path planning algorithm that can function in the Motion Planning Module of the AutoNav framework, irrespective of what is in the Conflict Detection and Mapping Modules of the framework.

By looking at the RRT* and the PRM, the following evaluations and considerations are made.

- Unlike the PRM, an initial roadmap of the map will not be created and only random sampling of the environment will be done. This removes the restriction of the map representation on the algorithm.
- The improved algorithm will use a function similar to the ExtendPRM function that only searches the first $n$ number of sorted points in the $tree_{PRM}$ to connect to the random sample. This was found to produce the cheapest path the quickest.
- The ExtendRRT* function is limited by the distance $\eta$ it can extend from the $tree_{RRT^*}$ and was found to restrict the possible connections to a sample point, but reduces the execution time adding the sample to the tree. Although the RRT* was faster to add new samples to the tree, for a conservative value of $\eta$, the rate at which the tree grows is slower compared to the PRM algorithm.
- Neither the PRM nor the RRT* implements a method to force the path to become more optimal, and this is addressed in the improved algorithm.

- The improved algorithm implements a library of manoeuvres similar to the PRM and RRT* algorithms, but combines the properties of STEER and LPM functions, as mentioned in Chapter 4. The improved algorithm tries to reach the end point precisely (similar to LPM) but ignores the end heading (similar to STEER). For this project it was chosen that reaching the end point precisely is more important than in what orientation the vehicle reaches the end point.

Now that a motion planning algorithm is found to meet the requirements of the problem statement, the adaptation and improvement of the algorithm is discussed in the next chapter.

# Chapter 7

# Implementation and Methodology

This chapter aims to provide a summary of the autonomous navigation system that was implemented on the test vehicle and used to demonstrate the motion planning algorithm developed. It starts by discussing the maps used to do planning on and how the Mapping Module of the AutoNav architecture was simplified. The implemented motion planning algorithm is discussed at length, comparing the differences to the motion planning algorithms in Chapter 6. The limitations of the application in this project is mentioned and the test procedures are set out.

## 7.1 Mapped environments

The sensor that enables the vehicle to "see" the surroundings and which is responsible for updating the information in the Mapping Module, has not been implemented for the environments considered for this project. The implementation of the Mapping Module is also not in the scope of the project, as set out in Section 2.4. The vehicle was therefore unable to detect new or dynamic information to add to the map, for example a moving obstacle or a pot-hole. In order to test the path planning algorithm and the execution of the path by the vehicle controller, all the information of the surrounding area in which the vehicle is to be tested should be contained in the map. This map is preloaded in the place of the Mapping Module, and is a temporary solution which will be removed when a future project is required to develop the Mapping Module. By using the differential GPS unit to provide location information, the vehicle can keep track where it is in the map relative to the base-station antenna of the DGPS system. The maps that are created for the testing of the motion planning algorithm use global GPS coordinates retrieved from Google Maps to define the boundaries and all of the stationary obstacles in the area. The area inside the map boundaries and outside the obstacle areas is the free space in which the vehicle is allowed to drive. In the following section the data structures used to represent the map, boundaries and obstacles are discussed. Following the discussion of the data structures is a graphical representation of the maps provided to the path planner, and it is compared with Google Maps images of the area in which the field tests will be done.

### 7.1.1 Data structures

Before a graphical representation of the maps can be given, an understanding of the data structures used to contain the map information should be acquired. The **Map** data structure is used to distinguish between the maps of different areas used for field testing. The **Map** data structure, Table 7.1, contains an array of the **Boundary** data structures and an array of the **Obstacle** data structures.

Table 7.1: The **Map** data structure

| **Map** | |
|---|---|
| Boundary | An array of **Boundary** points defining the corners of the map |
| Obstacle | An array of **Obstacle** points defining all the obstacles in the map |

One of the **Boundary** points is chosen to be the origin of the inertial axis system, as defined in Section 3.1, which is also referred to as the map axis system. The ground-station DGPS antenna is placed at the origin boundary point to enable the vehicle coordinates to be also referenced from this point. The **Boundary** data structure, Table 7.2, is simply the North and East axis position of one of the boundary points in the map, and are connected by straight lines.

Table 7.2: The **Boundary** data structure

| **Boundary** | |
|---|---|
| N | North axis position in the map |
| E | East axis position in the map |

The **Obstacle** data structure, Table 7.3, is similar to a **Boundary** as it also contains the North and East axis position information, but also contains the radius of the specific obstacle to define the obstacle area as a circle.

Table 7.3: The **Obstacle** data structure

| **Obstacle** | |
|---|---|
| N | North axis position in the map |
| E | East axis position in the map |
| radius | The radius of the obstacle |

To summarise, each corner point of the **Map** is represented by a **Boundary** point, with one of the boundaries located at the origin of the inertial axis. The base-station DGPS antenna is set up at this boundary point to provide the reference point. All areas in the **Map** in which the vehicle is not allowed to drive are represented by multiple circular **Obstacles**. The obstacles are defined as circles to simplify the conflict detection method, as introduced in Section 5.2.1, where the conflict region around the obstacle is determined by the conflict detection algorithm.

### 7.1.2 Maps

The graphical representation of the **Maps** used for field testing is provided below. The **Boundary** points form the corners of the map, connected by the blue straight line segments. The **Obstacles** are presented by the red circles. The three different maps were chosen for their different types of terrain to be able to evaluate the performance of the vehicle controller on more than one type of terrain. Between the three different maps, the number and size of obstacles are also varied.

#### Tarmac parking area

An empty parking area was used to do a series of field tests on. As shown in Figure 7.1, the parking bays are represented as obstacles in the map, and the only drivable area is on the tarmac road in the parking area.

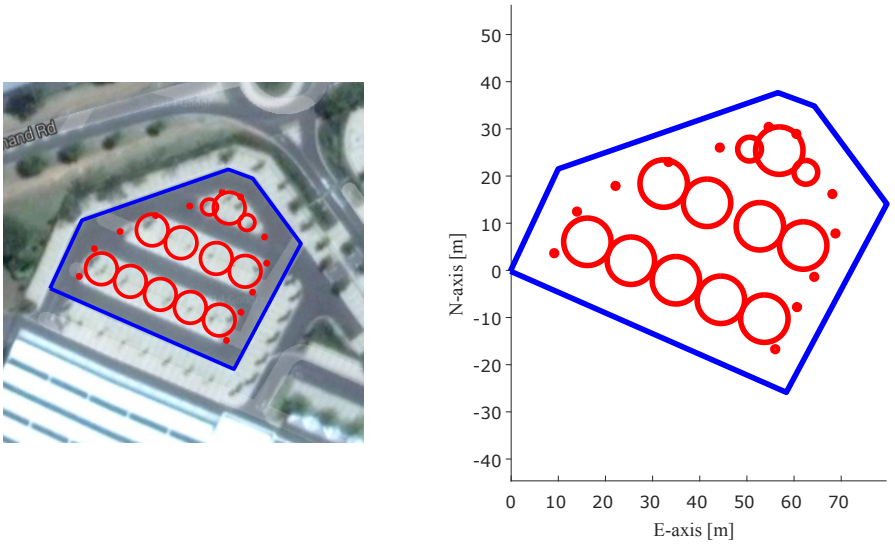



Figure 7.1: Comparison of the Google Maps image of the tarmac parking area and the generated map used by the path planner

### Gravel area

An empty gravel area was used to do another series of field tests on. As seen in Figure 7.2, there are no real obstacles in the area, and one large virtual obstacle was placed in the map to limit the drivable area in the map provided to the path planner.

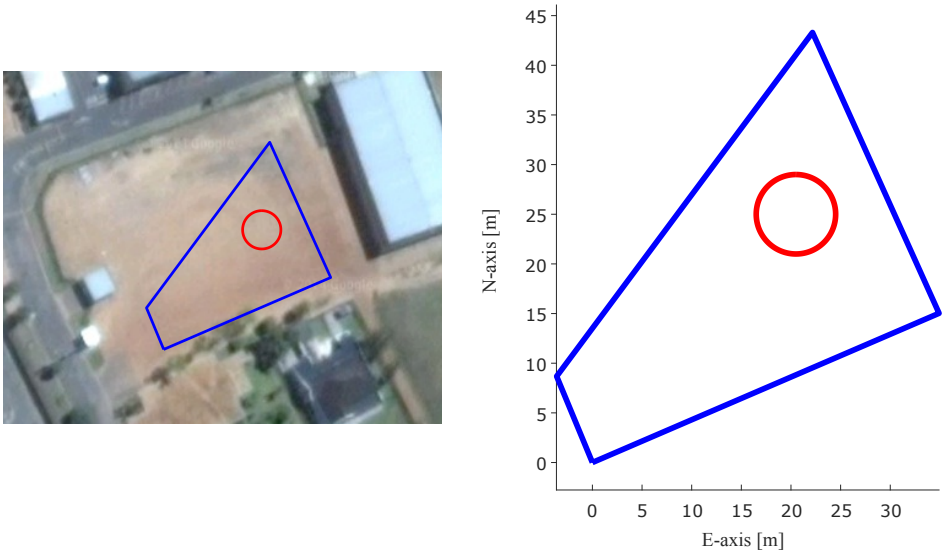



Figure 7.2: Comparison of the Google Maps image of the gravel area and the generated map used by the path planner

### Grass field

A small grass field was used to do the last series of field tests on. As seen in Figure 7.3, multiple small virtual obstacles were placed in the map to limit the drivable area in the map provided to the path planner.

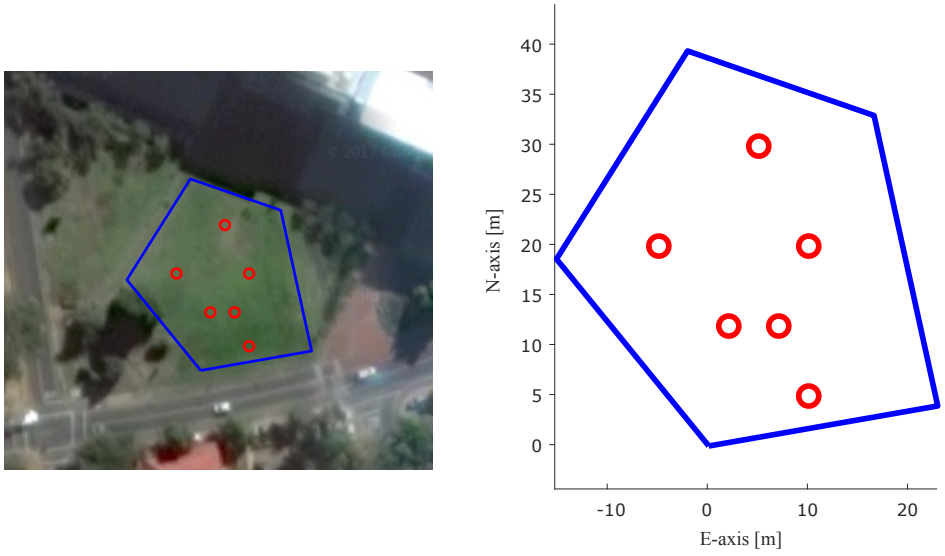



Figure 7.3: Comparison of the Google Maps image of the grass field and the generated map used by the path planner

## 7.2 Path planner

The path planning algorithm implemented in this project is an adaptation from the classic RRT and RRT* algorithms, discussed in Chapter 6. The first way in which the implemented algorithm differs from the classic RRT algorithms is by how the algorithms consider to add a randomly sampled point to the existing tree. The classic algorithm only tries to connect the nearest node to the sampled point, whereas the adapted algorithm tries to connect all the nodes in the tree to the newly sample point. The second difference is that the adapted algorithm makes use of a function to trim the tree regularly to remove nodes that would definitely result in a less optimal path than the latest one found. The classic algorithm does not implement such a feature. Another difference between the classic and adapted algorithm is that the classic algorithm does not guarantee that for every path found, it is more optimal than the previous one found. It also does not have the ability to implicitly improve on paths found. The adapted algorithm improves on every path found and guarantees a more optimal path, the longer it is allowed to run, in the following three ways:

- Connections to a newly sampled random point are first sorted from the lowest to the highest cost, before testing the connection for conflict. The first conflict-free connection will then be the cheapest too.
- The search area from which randomly sampled points are taken is restricted by a certain shape.
- The search area becomes smaller every time that a shorter path is found.

### 7.2.1 Data structures

Before the path planning algorithm can be discussed, it is necessary to gain an understanding of the data structures and notations used in the algorithm. With the knowledge of the data structures, it is easier to visualise what is mentioned in the algorithm. The RRT algorithm generates a tree of paths between a starting point $s_{start}$ towards a goal point $s_{goal}$. The root of the *tree* starts at $s_{start}$, with branches flowing out towards $s_{goal}$. The splitting point in a branch is represented by a **node**. The tips of the branches, where no further splitting has occurred, is referred to as leaf **nodes**. In Table 7.4, the first data structure, **node**, is given with a description of each element within.

Table 7.4: The **node** data structure

| **node** | |
|---|---|
| *location* | N and E coordinates in the map axis system |
| **manoeuvre** | The **manoeuvre** used to reach this node |
| *added_cost* | Accumulated cost from $s_{start}$ to this node |
| *projected_cost* | *added_cost* of this node plus the manoeuvre length to reach $s_{goal}$ from this node |
| *children* | All nodes that can be reached from this node |

From the **node** data structure, the following data structure is a **manoeuvre**, presented in Table 7.5.

Table 7.5: The **manoeuvre** data structure

| **manoeuvre** | |
|---|---|
| *Turn* | Type of manoeuvre used from the manoeuvre library. In this case it is distinguished between 'Left' turn, 'Right' turn, 'Straight' line or 'None' if no manoeuvre is possible |
| *p_start* | The starting **point** of the manoeuvre |
| *p_stop* | The end **point** of the manoeuvre |
| *length* | The path length between p_start and p_stop |

From the **manoeuvre** data structure, the **point** data structure is introduced, shown in Table 7.6. The **point** contains the minimum number of calculated vehicle states needed to do local planning between two points, discussed in Chapter 4.

Table 7.6: The **point** data structure

| **point** | |
|---|---|
| N | North axis position in the map |
| E | East axis position in the map |
| heading | heading angle of the vehicle in the map axis system |
| steer angle | vehicle steering wheel angle in the map axis system |
| vehicle speed | vehicle speed in the map axis system |

In this project, to facilitate the storage of **nodes** and searching through them, two lists of **nodes** are used. The first list, **paths**, is used to store all of the end **nodes** with *location* at $s_{goal}$ that resulted in a successful, conflict free path between $s_{start}$ and $s_{goal}$. These end **nodes** do not form part of the tree, but is necessary to represent the paths found. The data structure of **paths** is given in Table 7.7. The second list, **list_of_nodes**, is used to store all the **nodes** that form part of the tree, and is presented in Table 7.8.

Table 7.7: The **paths** data structure

| **paths** | |
|---|---|
| **nodes** | An array of end **nodes** resulting in paths found between $s_{start}$ and $s_{goal}$. This list is sorted in an ascending order according to the *added_cost* of the end **nodes** |

Table 7.8: The **list_of_nodes** data structure

| **list_of_nodes** | |
|---|---|
| node | An array of all **nodes** in the tree of paths, excluding end **nodes** listed in **paths** |

Finally, the random samples taken from the search space during path planning only contain location information and not a randomly sampled orientation. The **random sample** is presented in Table 7.9.

Table 7.9: The **random sample** data structure

| **random sample** | |
|---|---|
| N | North axis position in the map |
| E | East axis position in the map |

The *tree*, referred to in this chapter, is the linked list data structure definition of the RRT algorithms. It is a **node** that has no parent **node** and of which the *location* attribute is $s_{start}$. It is therefore the start of the tree of paths that aim to connect $s_{start}$ to $s_{goal}$. In this project, due to the accessibility of the parent and child **nodes** from both sides[1], the *tree* is simply kept as a representation of the tree structure. Whenever a specific node is required, it is searched for in the **list_of_nodes** by using a known attribute, like *location*, and not by stepping through each parent-child relationship.

In this project, the motion planning, conflict detection and mapping modules were implemented in MATLAB on the on-board laptop of the test vehicle. Due to this reason, in MATLAB, the **nodes** were kept in an array for ease of accessibility and storage management. If the path planner is ported to C or C++, an array of pointers to the

[1] **node.manoeuvre**.*p_start* points to the *location* of the parent **node** and **node**.*children* points to the *location* of the child **nodes**

address locations of the **nodes** will be sufficient. In the MATLAB program the parent **node** lists its children by containing the children **nodes'** location coordinate information. In the C or C++ version of the program the children **nodes** can be identified with pointers to their memory address locations.

To illustrate the data structures and their differences, an example of the tree of paths generated by the path planning algorithm is provided in Figure 7.4. The starting **node** *tree* is the starting point $s_{start}$, located at the origin with heading parallel to the North-axis. The **nodes** $n_1$ to $n_8$ are **random samples** connected to the *tree*, with coordinates as indicated in the figure.

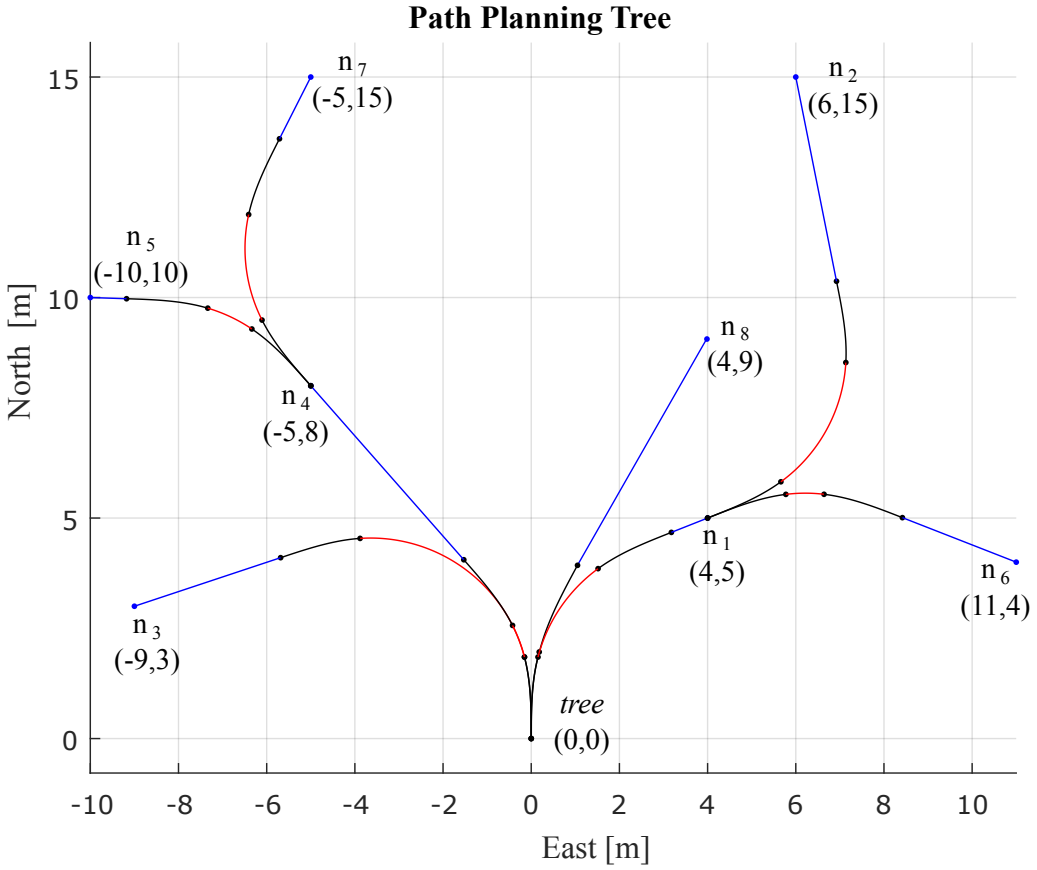



Figure 7.4: A representation of the tree of paths generated by the path planning algorithm

From Figure 7.4, the **list_of_nodes** data structure would be the array of nodes, including *tree*, sequentially added in the order that they were sampled:

$$\textbf{list\_of\_nodes} = \left\{ tree \quad n_1 \quad n_2 \quad n_3 \quad n_4 \quad n_5 \quad n_6 \quad n_7 \quad n_8 \right\}.$$

The **node** data structure of *tree* will take the form of

$$tree = \begin{Bmatrix} location \\ \textbf{manoeuvre} \\ added\_cost \\ projected\_cost \\ children \end{Bmatrix} = \begin{Bmatrix} \begin{Bmatrix} 0 & 0 \end{Bmatrix} \\ empty \\ empty \\ projected\_cost(s_{start} \rightarrow s_{goal}) \\ \begin{Bmatrix} n_1 & n_3 & n_4 & n_8 \end{Bmatrix} \end{Bmatrix}.$$

An example of one of the **nodes** $n_1$ would be

$$n_1 = \begin{Bmatrix} location \\ \textbf{manoeuvre} \\ added\_cost \\ projected\_cost \\ children \end{Bmatrix} = \begin{Bmatrix} \begin{Bmatrix} 4 & 5 \end{Bmatrix} \\ \textbf{manoeuvre}(tree \rightarrow n_1) \\ \textbf{manoeuvre}(tree \rightarrow n_1).length \\ projected\_cost(n_1 \rightarrow s_{goal}) \\ \begin{Bmatrix} n_2 & n_6 \end{Bmatrix} \end{Bmatrix}.$$

The **manoeuvre** between *tree* and $n_1$ can be represented by

$$\textbf{manoeuvre}(tree \rightarrow n_1) = \begin{Bmatrix} Turn \\ p\_start \\ p\_stop \\ length \end{Bmatrix} = \begin{Bmatrix} 'Right' \\ tree.location \\ n_1.location \\ \text{path length of } tree \rightarrow n_1 \end{Bmatrix}.$$

This concludes the data structures used in the path planning algorithm, which follows in the next subsection.

### 7.2.2 Path planning algorithm

With the necessary data structures defined to represent the information in the path planning algorithm, the discussion of the algorithm can be presented. The main loop of the path planning algorithm is presented first in this subsection, and follows with a discussion of each sub-function called from the main function.

### Main function

The main loop of the path planning algorithm is split over two pages and presented in Algorithm 10 which continues onto Algorithm 11.

**Algorithm 10** PathPlanner - Part 1

**function** Main( )
    set the number of random samples $N$
    set safety factor $SF$ for vehicle clearance to do conflict detection
    set the end goal **point** $s_{goal}$
    retrieve current vehicle states and store as starting **point** $s_{start}$
    initialise: $MAP$, generate transition curves, $Cost$, $tree$, empty **list_of_nodes**

    first try to connect $s_{goal}$ to $s_{start}$ with **manoeuvre** = LPM($s_{start}, s_{goal}$)
    check for conflict on the returned **manoeuvre** with
    $conflict$ = DetectConflict(**manoeuvre**, $SF, MAP$)
    **if** no $conflict$ **then**
        $end\_node$ = CreateNode(**manoeuvre**, **list_of_nodes**)
        add the $end\_node$ to the list of **paths** found between $s_{start}$ and $s_{goal}$
        update the parent node $s_{start}$ to point to its new child $s_{goal}$ with
        **list_of_nodes** = UpdateParent($end\_node$, **list_of_nodes**)
        update the $tree$ structure with $tree$ = UpdateTree($tree$, **list_of_nodes**)
        update the sampling area criteria to search within for a more optimal path
        $cost\_max$ = UpdateSearchCriteria($cost\_max, tree$, **paths**)
    **end if**

    **for** $n = 1 : N$ **do**
        sample a random point in the search space with
        $p\_sample$ = SampleSearchSpace($s_{start}, s_{goal}, cost\_max$)
        try to connect all the **nodes** in **list_of_nodes** to the sampled point $p\_sample$

    **temp_list_of_nodes** = ConnectToSamplePoint($p\_sample$, **list_of_nodes**)
        for multiple connections to the sampled point, first sort according to cost, from
        cheapest **temp_list_of_nodes** = SortCost(**temp_list_of_nodes**)
        test for conflict, starting from cheapest, and return the first conflict free
        connection to $p\_sample$ with
        **node_p_sample** = TestSample(**temp_list_of_nodes**, $SF, MAP$)

**Algorithm 11** PATHPLANNER - Part 2

21: **if** a conflict free path was returned and **node_p_sample** is not empty **then**
22: try to connect **node_p_sample** to the end point $s_{goal}$ with **manoeuvre** = LPM(**node_p_sample**, $s_{goal}$)
23: add the **manoeuvre** cost to the accumulated cost of **node_p_sample** **node_p_sample**.$projected\_cost$ = **node_p_sample**.$added\_cost$+ **manoeuvre**.$length$
24: save **node_p_sample** into the **list_of_nodes** using **list_of_nodes** = ADDNODE(**node_p_sample**, **list_of_nodes**)
25: update the parent node of **node_p_sample** **list_of_nodes** = UPDATEPARENT(**node_p_sample**, **list_of_nodes**)
26: update the *tree* structure with *tree* = UPDATETREE(*tree*, **list_of_nodes**)
27: test for conflict between **node_p_sample** and end point $s_{goal}$ $conflict$ = DETECTCONFLICT(**manoeuvre**, $SF$, $MAP$)
28: **if** no $conflict$ **then**
29: $end\_node$ = CREATENODE(**manoeuvre**, **list_of_nodes**)
30: add the $end\_node$ to the list of **paths** found between $s_{start}$ and $s_{goal}$
31: update the parent **node_p_sample** to point to it's new child $s_{goal}$ **list_of_nodes** = UPDATEPARENT($end\_node$, **list_of_nodes**)
32: update the sampling area criteria to search for a more optimal path $cost\_max$ = UPDATESEARCHCRITERIA($cost\_max$, $tree$, **paths**)
33: **if** $end\_node.added\_cost < trim\_criteria$ **then**
if a shorter path has been found, update the criteria for trimming
34: $trim\_criteria = end\_node.added\_cost$
35: **end if**
36: set $trim\_flag = 1$ to trim the the *tree* of all **nodes** with $projected\_cost > trim\_criteria$
37: **end if**
38: **end if**
39: **if** $trim\_flag$ is *true* **then**
40: recursively remove all leaf **nodes** from *tree* using $trim\_criteria$ [**list_of_nodes**, $remove\_child$] = REMOVELEAFNODE($tree$, $trim\_criteria$, **list_of_nodes**)
41: reset trim flag $trim\_flag = 0$
42: **end if**
43: **end for**
44: **end function**

The user inputs can be defined in lines 2 to 4 and include the number of random samples

to test in the environment, the clearance safety factor of the vehicle to be used in the conflict detection test, and the goal that the vehicle has to reach. Lines 7 to 15 checks whether there is a conflict-free path directly from the start to the goal. For the number of random samples specified, lines 16 to 43 are repeated. With each repetition a random sample is taken and a temporary connection to every node in the tree is created and sorted according to the cost, lines 17 to 19. In lines 20 to 26, each temporary connection is tested for conflict, and the first conflict free connection is returned and added to the tree. In lines 27 to 31, the newly added connection has a heading at its endpoint which is used to try and connect it to the goal, tested for conflict, and if no conflict is found, a path is found and created between the start and goal. In lines 32 to 42 the length of the path is compared to previous paths found. If this new path is shorter, the search space is reduced and the nodes in the tree that are now outside the search space are removed by trimming.

### Sub-functions

Starting from the top of Algorithm 10 and continuing through to the end of Algorithm 11, the sub-functions used in the main loop is listed below.

LPM(): The sub-function LPM() is discussed in Chapter 4 in Algorithm 1, where the local planning method was first introduced. It takes two points to be connected, for example a node in the tree and a randomly sampled point, and the manoeuvre library as inputs. Using the manoeuvre library, the function builds a path between the two points and returns it to the main function.

DetectConflict(): For this project, the conflict detection algorithm was not the focus, and therefore simplified to suite the needs of the motion planning algorithm implemented. The conflict detection function takes a manoeuvre, safety factor and the map of the environment as inputs. It then calculates a trajectory from the information in the **manoeuvre** with a specified step size resolution. The interval length between the points in the trajectory was chosen to be not too fine (to save on calculation time), but fine enough to ensure that an obstacle would not fit in between points of the trajectory. The function then cycles through the **Boundary** points in the **Map** and determines if every trajectory point generated from the **manoeuvre** lies within the boundaries. If any of the trajectory elements are outside of the map boundaries, the function stops and returns a *true* condition for conflict detected. If the trajectory lies completely within the boundaries of the map, the function continues to cycle through the **Obstacles** of the map and tests each point of the trajectory to determine if it is outside of the obstacle area. If any of the trajectory points are inside of an obstacle, the function stops and returns a *true* condition for conflict detected. The area of the obstacle is expanded by the safety factor (an input

of the function) and the area of the vehicle, by means of Minkowski addition, as discussed in Section 5.2.1. If no conflict was detected, the function returns a $false$ condition.

CreateNode(): This function is used to create a **node** data structure at the end point of a **manoeuvre** that was tested to be free of conflict. The function populates the data elements in a **node** by using information from the **manoeuvre**. By using the starting point of the manoeuvre, it searches the **list_of_nodes** for the index of the parent node. The node to be created then receives the following assignments:

$$\begin{aligned} parent\_node\_added\_cost &= \textbf{list\_of\_nodes}(index\_parent).added\_cost \\ \textbf{node}.location &= \textbf{manoeuvre}.p\_stop \\ \textbf{node}.manoeuvre &= \textbf{manoeuvre} \\ \textbf{node}.added\_cost &= parent\_node\_added\_cost + \textbf{manoeuvre}.length \\ \textbf{node}.projected\_cost &= empty \\ \textbf{node}.children &= empty \end{aligned}$$

UpdateParent(): This function is used to update the parent **node** of a newly created child **node** to contain a reference to its new child. The function searches the **list_of_nodes** for the parent **node** that has a *location* corresponding to the child **node**'s **manoeuvre**.*p_start* property. Once the parent **node** is found, the *location* of the child **node** is appended to the parent's **node**.*children* attribute.

UpdateTree(): The UpdateTree() function is similar to the UpdateParent() function, except that it only updates the top-most parent **node** whenever a new **node** is connected to it as a child.

UpdateSearchCriteria(): This function starts by searching through **paths** for the end **node** with the lowest accumulated cost. If a **node** is found with *added_cost* less than the current cost value, the cost criteria is updated. The cost criteria is used to define the size of the sampling area within which the path planner searches for paths between $s_{start}$ and $s_{goal}$.

SampleSearchSpace(): This function samples the search space for random candidate points to be connected to the *tree*. As mentioned in the discussion of the data structures, the **random sample** only contains location information in the search space, and not any orientation information. The search space is therefore an area in the two-dimentional map of the environment within which a path has to be found. For this project, it was chosen to represent the area of the search space with an ellipse that is a cost function of the path length. As shown in Figure 7.5, an ellipse with centre at the origin, has the property that the length $c$ from a focal point, $F_1$, to the edge of the ellipse and back to the second focal point, $F_2$, is the same in all directions. This relates to $c_1 = c_2 = c_3$. By having $F_1$ and $F_2$

correspond to the start and end goal, and by defining the maximum length of $c$ to be the accumulated cost of a path found, the search space becomes a function of the cost of a path. The path with differential constraints included will always be longer than the straight line approximation thereof. Thus, the boundary of the ellipse with $c_{max} = added\_cost$ of an end **node** will always enclose the path to reach the respective end **node**. The value of $c_{max}$ is reduced by UpdateSearchCriteria() every time that a shorter path has been found. This ensures that no shorter path lies outside the ellipse, and it is unnecessary to sample outside the ellipse. The uniformly sampling of an ellipse is described with reference to Figure 7.5 below.

The uniform sampling function starts by calculating the length $f$ of a straight line between $s_{start}$ and $s_{goal}$. It then places $F_1$ and $F_2$ on the x-axis, separated by distance $f$, centred around the y-axis. The values of $a$ and $b$ are given by

$$a = \frac{c_{max}}{2} \quad \text{and} \quad a > \frac{f}{2}$$
$$b = \sqrt{a^2 - \left(\frac{f}{2}\right)^2}$$

The procedure to sample an ellipse then becomes:

1. Begin with a unit circle:

$$\begin{aligned} \text{Sample a random radius in unit circle}[0,1] &\rightarrow \sqrt{r_{random}} \\ \text{Sample a random angle in unit circle}[0,2\pi] &\rightarrow \theta_{random} \\ \implies x_r &= r_{random}\cos(\theta_{random}) \\ y_r &= r_{random}\sin(\theta_{random}) \end{aligned}$$

2. Stretch the circle to fit the ellipse:

$$\begin{aligned} x &= ax_r \\ y &= by_r \end{aligned}$$

3. Rotate and translate to align $F_1$ with $s_{start}$ and $F_2$ with $s_{goal}$ using:

$$\text{Rotation matrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

$$\text{Translation matrix} = \begin{bmatrix} x_T \\ y_T \end{bmatrix}$$

with $x_T$ and $y_T$ the translation lengths.

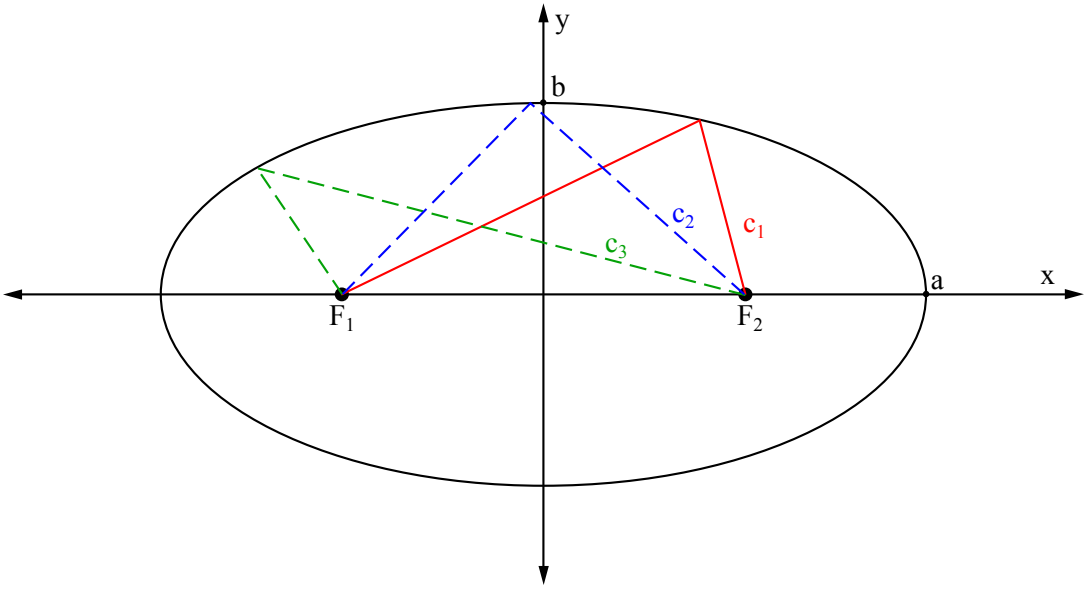



Figure 7.5: Defining the variables used to produce an elliptic 2-D sampling space

ConnectToSamplePoint(): This function cycles through all **nodes** in **list_of_nodes** and calls the LPM() function for each to try and connect it to the sampled point *p_sample*. For each successful connection, the function creates a **node** at the sampled point and adds it to a temporary list. The function then returns this temporary list of **nodes** at *p_sample*.

SortCost(): The temporary list of **nodes** returned from the ConnectToSamplePoint() function is sorted in an ascending order according to each **node**'s accumulated cost. This results in a temporary list of **nodes** that is sorted from the cheapest to the most expensive.

TestSample(): Each **node** in the sorted temporary list of **nodes** returned from SortCost() is then tested for conflict using DetectConflict(). The function TestSample() returns the first conflict free (and will also be the cheapest) connection to *p_sample* in the form of a **node** from the temporary list of **nodes** at *p_sample*.

AddNode(): This function searches through the **list_of_nodes** for an empty space to add a new **node** to, or appends the new **node** to the **list_of_nodes**.

RemoveLeafNode(): This is a recursive function, calling itself to step through the *tree* according to the parent-child relation to reach a **node** that has no *children* nodes. The **node** with no *children* is called a leaf **node**. The purpose of the function RemoveLeafNode() is to trim the *tree*. The *tree* is trimmed by removing all the leaf **nodes** of which the *added_cost* is already more than the cost of a path found between $s_{start}$ and $s_{goal}$. This reduces the number of **nodes** in the tree, and ensures that calculation time will not be spent on finding paths that are less optimal. With the availability of

**list_of_nodes**, this function could have been optimised to search each node in the **list_of_nodes** for a node with an empty *children* attribute. However, it was decided to have at least one function that behaves similarly to the RRT planning method and prove that a **node** at the end of a branch can be found by traversing through the parent-child relation from the root **node**.

With the data structures of the path planner defined, and the path planning algorithm described, the limitations of implementing the complete system can be discussed.

## 7.3 Limitations

The main goal of the project is to develop and demonstrate a path planning algorithm for a terrestrial vehicle. A summary of the implemented limitations and assumptions made for the demonstration are listed below.

From Chapter 3 it was found that the vehicle controller can acceptably achieve a constant driving speed of 3 m/s, a steering angle rate of 0.5 rad/s and a maximum steer angle of 0.2 radians. These values were fixed, and implemented to limit the manoeuvres available to the path planner. Therefore, the path planner always produces a path with a constant reference vehicle speed, fixed length and curve of the transition segments into and out of a constant curvature turning circle. The lengths of the constant curve circle arcs and the straight line segments are variable. The vehicle controller was also kept the same across the three different types of terrain, including the cross-track error controller. The cross-track error controller was the last control feature to be developed, and due to time constraints of the project, after it was implemented on the vehicle, it was not fine-tuned after the first few field tests were done. Lastly, the gear selector could not be operated by the vehicle controller due to one of the actuator control boards that was damaged. It was left to the human rider present on the test vehicle during autonomous tests to ensure the vehicle was set in the 'Forward' gear.

The Mapping, Conflict Detection and Motion Planning Modules are implemented in MATLAB on the on-board vehicle laptop. Again, due to time constraints of the project, these modules were not converted to C++ to be implemented as part of the laptop GUI which interfaces directly with the on-board controller of the vehicle. Therefore, to successfully demonstrate the concept, efficient implementation of the planning algorithm was not necessary.

The implemented path planning algorithm does not include the ability to do replanning in real-time. If replanning is required, the vehicle will have to stop and the replanning must be initiated by the user on the vehicle laptop. This will result in the path planner to discard the tree of paths found up to that point, and start over with a completely new tree with its root at the current position of the vehicle. The main motivation for not

implementing a real-time replanning capability was due to the amount of work required to adapt the interfaces between the MATLAB program, the C++ GUI and the vehicle controller implemented in C.

## 7.4 Test Procedure

For each of the three maps described earlier in the chapter, the following test procedures are repeated to acquire the field test results presented in Chapter 8.

1. Do pretest inspections of the vehicle (battery, fuel, tyre pressure, etc.)
2. Take the vehicle to the testing area.
3. Initialise the controllers and path planning algorithms, providing the necessary user inputs.
4. Start the path planning algorithm.
5. When a path is found, ensure everything is ready and the area is safe, and enable the autonomous navigation system.
6. Let the vehicle controller execute the planned path, only allowing the human rider to intervene in an emergency, which then results in the test to be discarded.
7. When the vehicle is stopped, Steps 3. to 6. can be repeated from the current position, or the vehicle can be moved manually to a desired starting position and orientation before repeating Steps steps 3. to 6.
8. After a determined number of successful paths planned and executed, the testing is finished.
9. The test results are retrieved from the on-board laptop and on-board controller to be analysed.

The inputs that the user had to provide to the path planning algorithm for executing a test include:

- Number of random samples
- Select the map to load
- Set the goal point to be reached
- Set the initial cost to include the whole map in the sampling area

It was found that 500 random samples were enough to repeatedly produce a path in all three different maps, and was kept constant for all the tests done. This usually resulted in a planning time of less than 50 seconds.

In this chapter the maps of the environment are chosen, defined and implemented in the place of the Mapping Module of the AutoNav framework. Three different maps, each with different terrain, will be used to test and demonstrate the path planning algorithm developed for this project. The implementation of the path planning algorithm is also discussed in this chapter, followed by the limitation of the demonstration that are to be expected. In the following chapter the test results for the three different maps are provided and discussed.

# Chapter 8

# Results and Analysis

The test results of the implemented system, as described by the previous chapter, are presented in this chapter. The performance of the vehicle controller is evaluated in Section 8.1 by using field test results. In Section 8.2, the performance of the path planning algorithm is demonstrated with simulation results.

## 8.1 Field Tests

The field test results acquired in the final stages of the project with the path planning algorithm implemented and vehicle controllers in place, are provided in this section. The different areas used for field testing was discussed in Section 7.1.2. Only one set of field results per area are provided to illustrate the performance of the vehicle controller. The tests results done in the tarmac parking area are given first, followed by the results of the gravel area and lastly the results of the grass field are presented.

In Figures 8.1, 8.3 and 8.5, the tracking results of the vehicle (blue) and the reference path (red) produced by the path planner are presented. The map boundary is defined by the blue polygon drawn around the track and the non-drivable areas are represented by the red circular obstacles. From the starting point indicated, the path planner was given the task to find a conflict free path to the finishing point, also indicated. The solution to the task (the optimal path found) is the reference path provided to the vehicle controller along with reference controller inputs, as indicated in Figures 8.2, 8.4 and 8.6, and include the vehicle speed and steering angle along the track. The engine speed reference shown in these figures mentioned last, is the inner-control loop reference from the vehicle speed controller, and is not provided by the path planning algorithm.

### 8.1.1 Tarmac

In Figure 8.1, the reference path and the path driven by the test vehicle is shown in the map of the tarmac parking area. From the indicated starting point, and comparing the vehicle states in Figure 8.2, the vehicle was accelerated from rest by the vehicle controller with a constant engine speed reference. Once the vehicle speed reaches 2.5 m/s, the vehicle speed controller took over and used the reference vehicle speed provided by the path planner. A slight left turn was encountered just after the acceleration stage of the vehicle, followed by a longer left turn around the 90° bend in the road. The reference path finished with a long straight stretch to the end point.

From Figure 8.1 it can be seen that the cross-track error controller compensated for the initial offset in the vehicle's heading to the heading of the reference path. For each turn in the path, the vehicle position overshoots the reference position and the cross-track error controller responds to correct the track offset. During the final straight stretch the settling time of the cross-track error controller can be seen to be rather slow and conservative, as was the initial design.

By looking at the steer angle reference over time in Figure 8.2, it can be seen that it is not smooth and contains high frequency information. This is the influence of the cross-track error calculator and controller input, which is added to the reference steer angle from the path planner. Nonetheless, the cross-track error controller produced good tracking ability of the reference path.

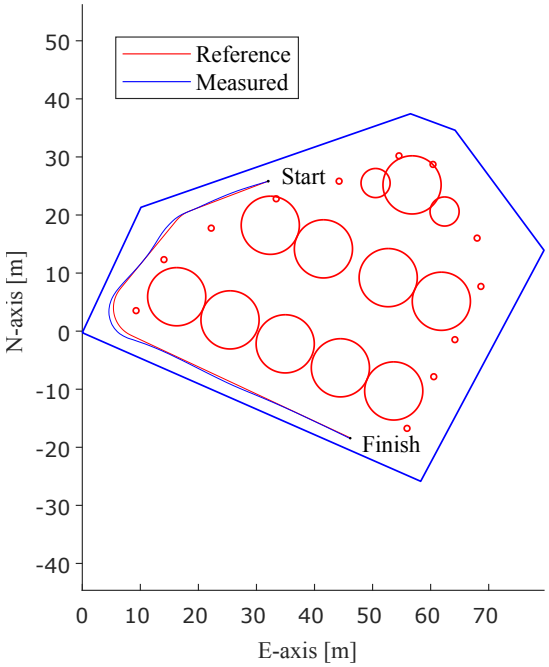



Figure 8.1: Field test results: reference and actual vehicle position in the tarmac parking area

From Figure 8.2, the inner-control loop reference provided by the vehicle speed controller to the engine speed controller is also shown and the corresponding throttle actuator command for the duration of the path.

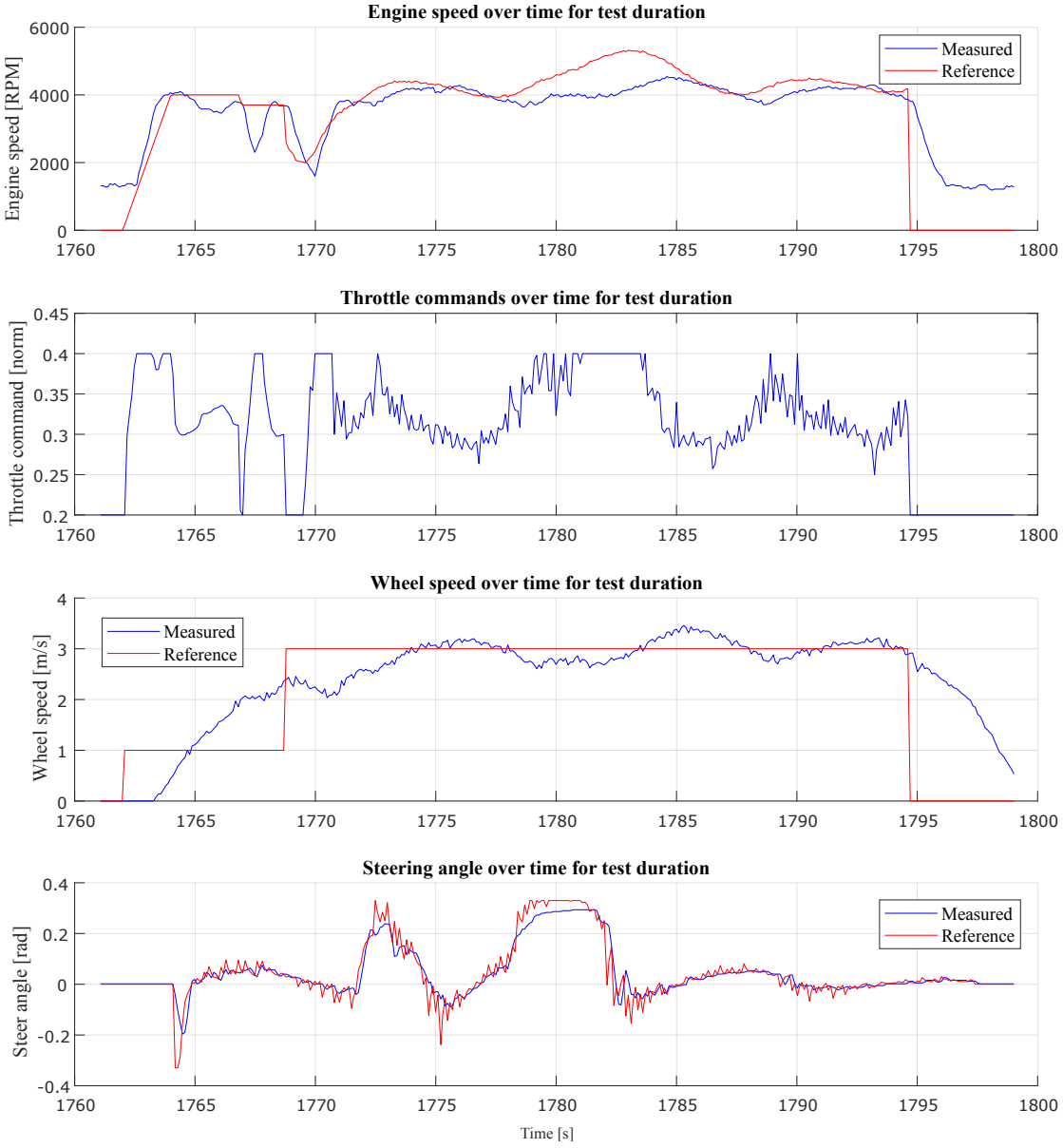



Figure 8.2: Field test results: reference and actual vehicle states during the test in the tarmac parking area

### 8.1.2 Gravel

In Figure 8.3, the reference path and the path driven by the test vehicle is shown in the map of the gravel area. From the indicated starting point, and comparing the vehicle states in Figure 8.4, the vehicle was accelerated from rest in the same way as described for the tarmac results. For this test, the reference path starts with a straight segment, followed by a long left turn and finishes with a straight stretch.

From Figure 8.4 it can be seen that during the long left turn, the steer angle reaches its limit as the cross-track error controller tries to compensated for the track offset. Also, the throttle command reaches its upper limited for the duration of the turn as the engine speed reference increases, but the actual engine speed stays relatively constant and only starts to increase at the end of the turn. From this it can be deduced that the engine is experiencing increased load during the turn due to the large steer angle and side slip of the front wheels.

By looking at the steer angle reference over time in Figure 8.4, the high frequency information noted for the test in the tarmac parking area is still present. This is due to the cross-track error calculator and controller that was not changed from the previous test discussed. It should be noted that on the tarmac the steering mechanism was damped more by the frictional force of the terrain than on the gravel and the high frequency effects from the cross-track error controller did not result in a high frequency response of the system. From the results in Figure 8.4 it can be seen that the measured steering angle contains a higher frequency response than in Figure 8.2.

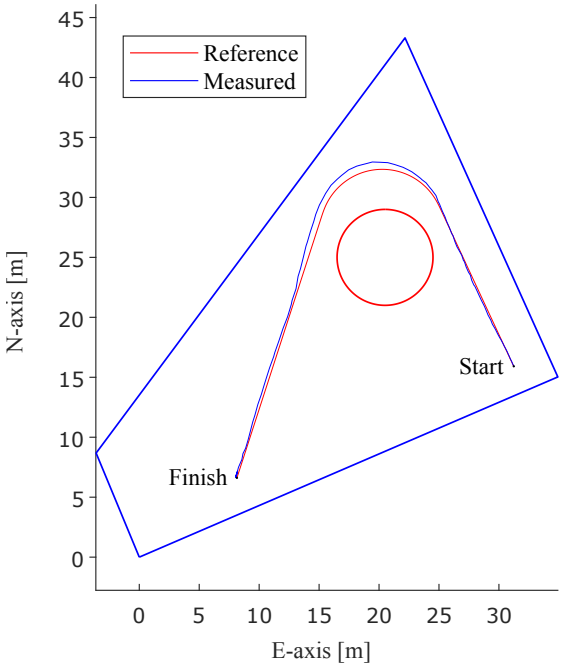



Figure 8.3: Field test results: reference and actual vehicle position on the gravel area

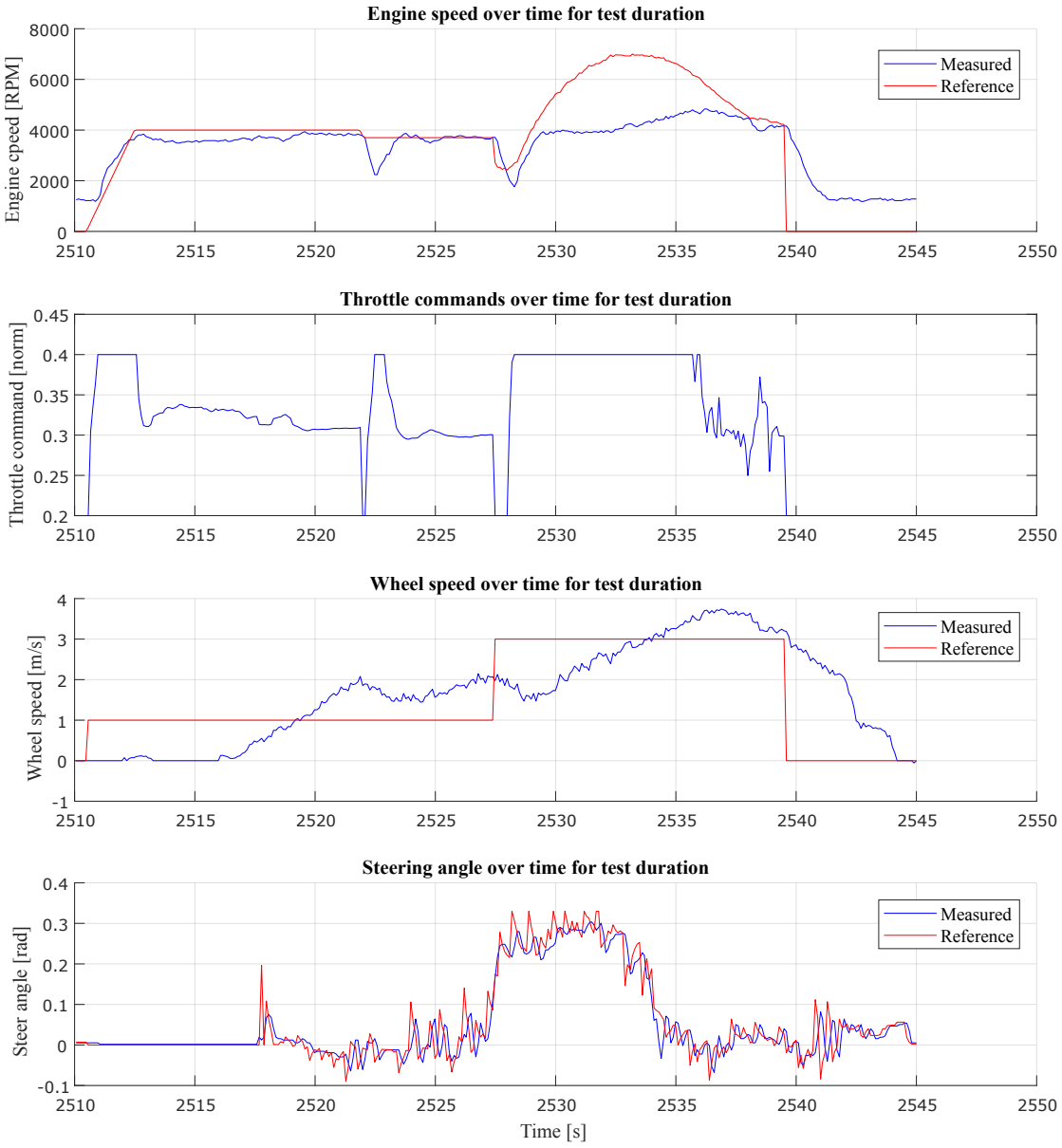



Figure 8.4: Field test results: reference and actual vehicle states during the test on the gravel area

### 8.1.3 Grass

In Figure 8.5, the reference path and the path driven by the test vehicle is shown in the map of the grass field. From the indicated starting point, and comparing the vehicle states in Figure 8.6, the vehicle was accelerated from rest in the same way as described for the previous two sets of results. For this test, the reference path starts with a straight segment, followed by a long left turn and a straight stretch before ending with another slight left turn just before the finishing point. It should be noted that this test area is much smaller than the previous two testing areas used.

From Figure 8.6 it can be seen that during the long left turn, the steer angle again reaches

its limit as the cross-track error controller tries to compensated for the track offset. The vehicle tended to slip more around corners when turning on the grass than on gravel or tarmac. In this case the throttle commands stayed within limits during the turns and from the wheel speed results the vehicle controller followed the reference vehicle speed rather well. It is however noted that the engine speed measurement stays constant at zero for the duration of the test. It is suspected that the engine speed sensor failed, as this had happened before during the testing and development stage of the vehicle controllers. The advantage of this failure is that it proves the redundancy of the control system.

By looking at the steer angle reference over time in Figure 8.6, the high frequency information noted for the previous two tests is still present. This is due to the cross-track error calculator and controller that was not changed from the previous tests discussed.

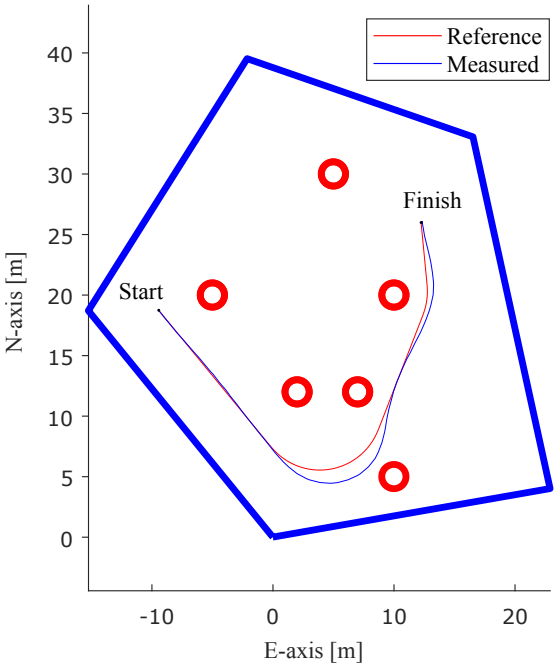



Figure 8.5: Field test results: reference and actual vehicle position on the grass field

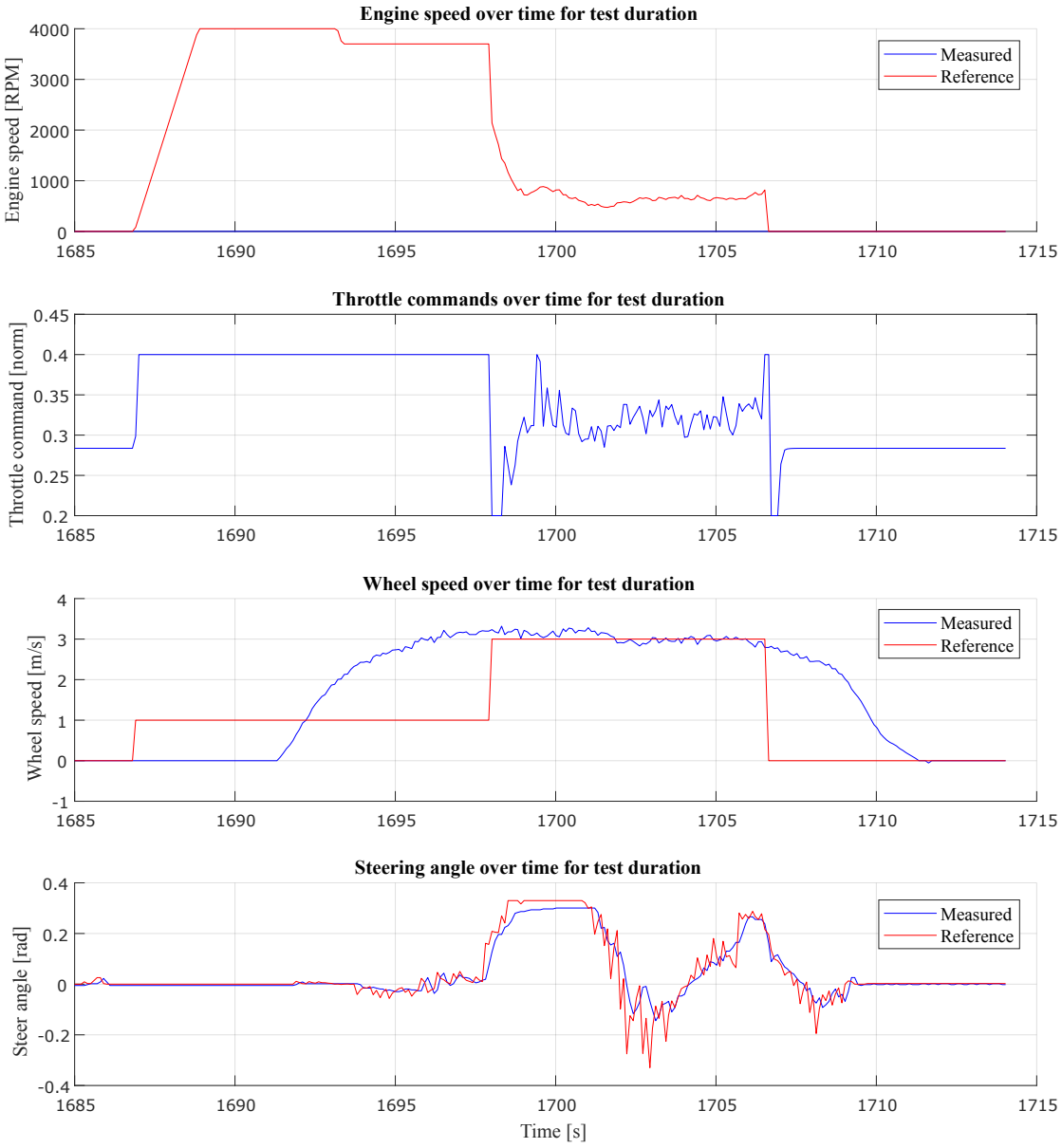



Figure 8.6: Field test results: reference and actual vehicle states during the test on the grass field

### 8.1.4 Conclusion of field tests

From the field test results, the vehicle controller did well to execute the reference path provided by the path planning algorithm, but improvements can be made in future work. The deviations from the reference path can be compensated by adjusting the kinetic and dynamic constraints in the path planner to better represent the actual capabilities of the vehicle seen in the field results. One of the main reasons for the deviations during turning is the side slip which is more apparent than what was initially assumed it would be. For an increase in the side slip at the front wheels, the steering load on the steering mechanism reduces and results in faster steering control and better tracking of reference steer angle. However, with the increase in side slip, the grip of the front wheels on the terrain reduces

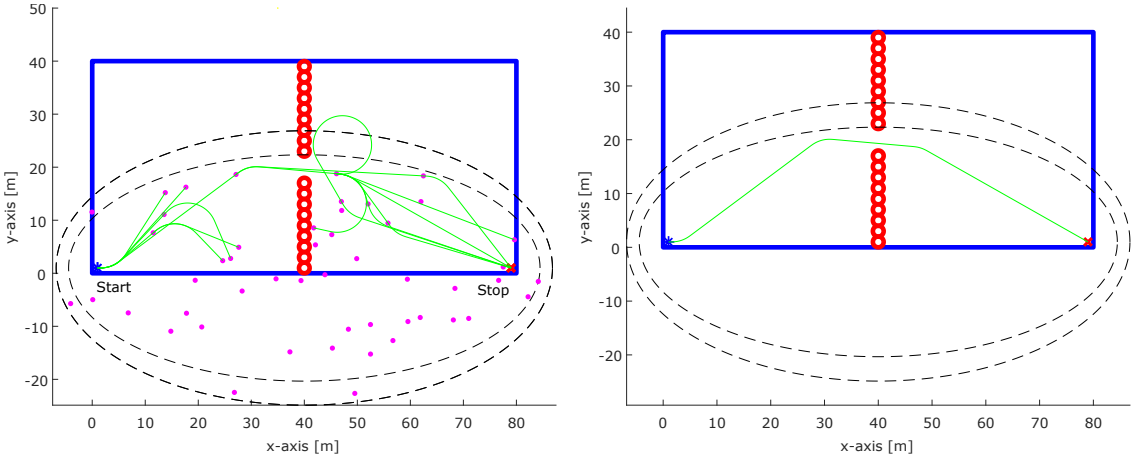
and results in the overshoot on the vehicle position control during a turn. The modelling of the relation between steer angle and vehicle trajectory should be improved.

## 8.2 Planning simulations

To measure the capability of the path planning algorithm, various simulated scenarios are provided to the algorithm. The resulting performance of the algorithm to produce an optimal path for each scenario can be compared to multiple iteration of each scenario and between different scenarios. A detailed performance analysis of the path planning algorithm was not done for this project. The following set of simulation results illustrate the capability of the algorithm, and suggest ways in which the performance can be analysed further.

### 8.2.1 Large open space with small gateway

The first scenario was chosen to represent the classic problem of a terrestrial vehicle that is in a room and needs to find a path through the door to the next room or outside area. On a larger scale, this could be an autonomous agricultural vehicle that is required to drive from one harvest field through a gate to the next field.




(a) Tree of paths (green) between start and stop points for 50 random samples (magenta)

(b) Shortest path (green) and the reduction in search space (from the larger black dashed ellipse to the smaller one)

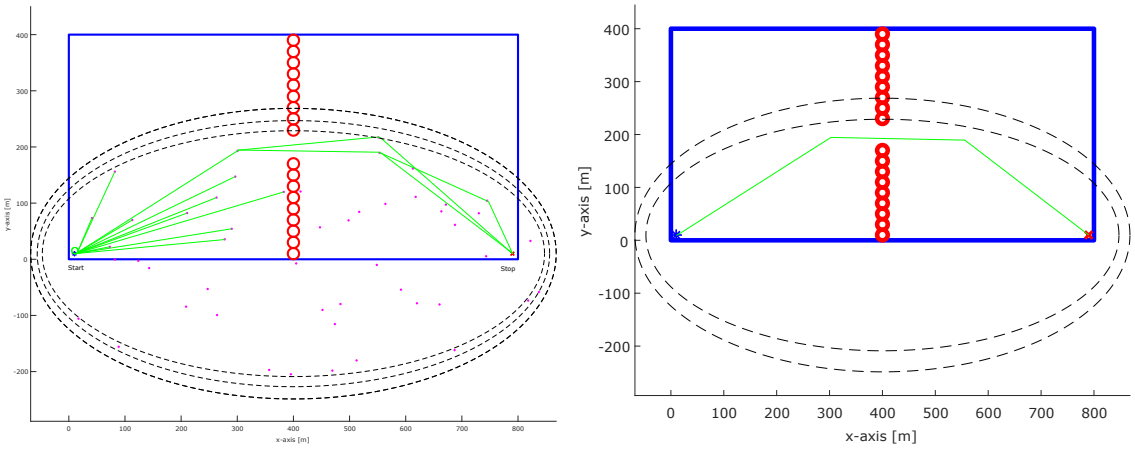
Figure 8.7: Simulation result of a large open area with a small gateway

The scenario of finding a path in a large open area with a small gateway was simulated, as indicated in Figures 8.7. The number of random samples were reduced to only 50 for the purpose of minimising the information in Figure 8.7(a), so that it is easier to see the tree of paths found. It was chosen to align the starting point and end point to the same

side of the map, but in opposite corners, with the gateway in die centre between the two empty areas. The starting point has a specified initial heading. Figure 8.7(a) provides the map with random samples (magenta), the tree of paths found (green) and the search space represented by an ellipse (black dashed lines) that became smaller for every shorter path found. Figure 8.7(b) reduces the information to only show the shortest path found from the starting point to the end point.

### 8.2.2 Comparing small and large maps

The scenario of finding a path in a large open area with a small gateway, mentioned in the previous section, was repeated with an up-scaled version of the map, as seen in Figure 8.8. The motivation for repeating the simulation on a larger version of one of the maps is to illustrate how the planner searches for a path in a very large area with the rest of the requirements staying constant. The vehicle constrains remain the same, and it is only the map that is up-scaled. Again, the number of random samples were reduced to only 50. Figure 8.8(a) provides the map with random samples, the tree of paths found and the search space represented by an ellipse that became smaller for every shorter path found. Figure 8.8(b) reduces the information to only show the shortest path found from the starting point to the end point. From Figure 8.8(b) it can be seen that the resulting path found after 50 samples takes a similar form to the shortest path found in Figure 8.7(b). It can thus be noted that the size of the map does not influence the path planning algorithm's ability to find a path.



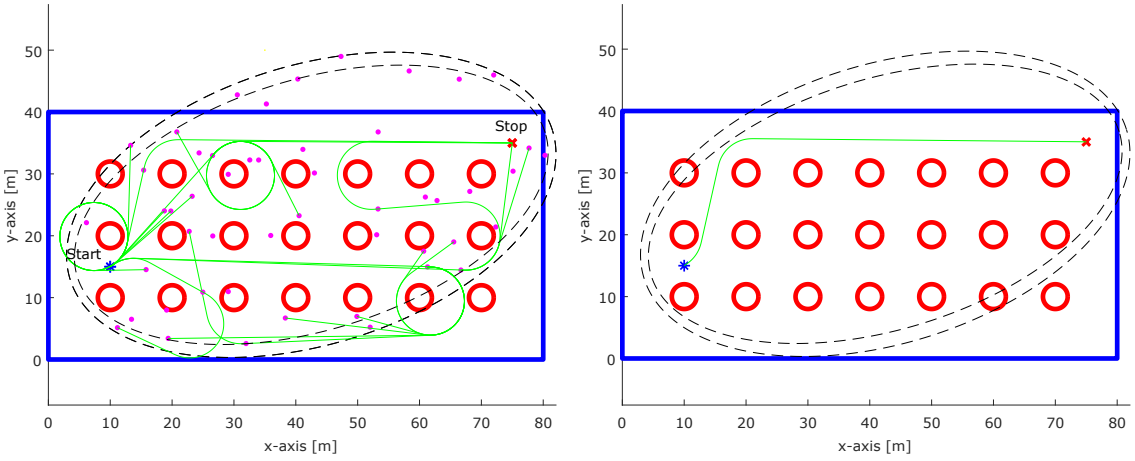

(a) Tree of paths (green) between start and stop points for 50 random samples (magenta)

(b) Shortest path (green) and the reduction in search space (black dashed ellipse)

Figure 8.8: Simulation result of an up-scaled large open area with a small gateway

### 8.2.3 Evenly spaced high density obstacles

To illustrate the flexibility of the the path that can be produced, an evenly spaced grid of obstacles was placed in the map. The starting point and end point was chosen at opposite ends of the map to force the algorithm to find a path through the grid of obstacles. The scenario of finding a path in an area populated by a high density of evenly spaced obstacles was simulated, as indicated in Figure 8.9. The number of random samples were reduced to only 50 for the purpose of reducing the information shown in Figure 8.9(a), so that it is easier to see the tree of paths found. Figure 8.9(a) provides the map with random samples, the tree of paths found and the search space represented by an ellipse that became smaller for every shorter path found. Figure 8.9(b) reduces the information to only show the shortest path found from the starting point to the end point. From Figure 8.9(a) it can be seen that the tree of paths contain a lot of turns and do not always extend towards the end point. This indicates that the algorithm is not limited to the direction in which it searches for a path and will not become trapped within local minima.




(a) Tree of paths (green) between start and stop points for 50 random samples (magenta)

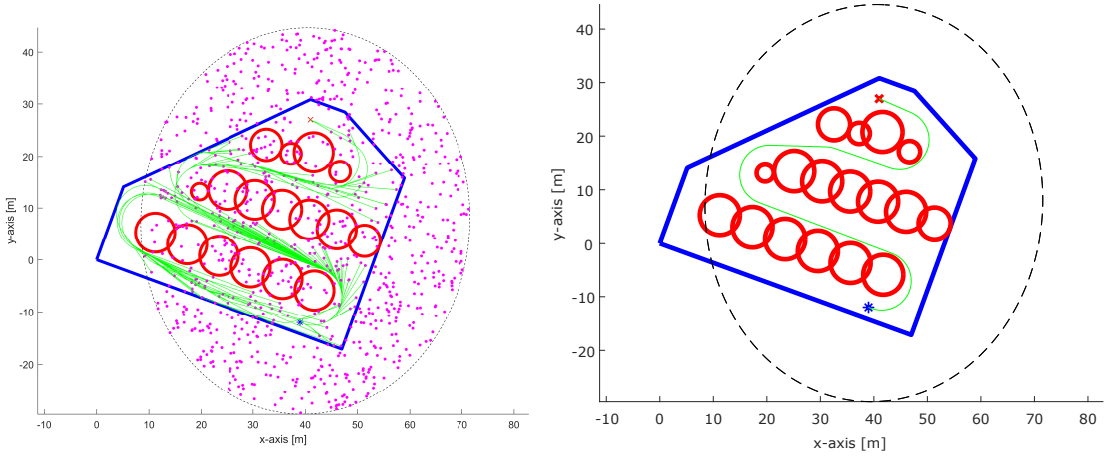
(b) Shortest path (green) and the reduction in search space (black dashed ellipse)

Figure 8.9: Simulation result of an area with evenly spaced obstacles

### 8.2.4 Long straight corridors

The scenario of finding a path in long straight corridors was simulated by using the map of the tarmac parking area and closing off some sections, as indicated in Figure 8.10. This could represent a scenario of a car with the ability to park itself to also navigate to an empty parking space within a parking area. It was necessary to increase the number of random samples to a 1000 to allow the algorithm to find a reasonable path, if any. Figure 8.10(a) provides the map with random samples, the tree of paths found and the search space represented by an ellipse. Figure 8.10(b) reduces the information to only provide the

shortest path found from the starting point to the end point. In Figure 8.10, the elliptic search area is large and extends beyond the map. This is due to the long and winding path length, of which the search space is a function. In Figure 8.10(a), it can be seen that a large portion of the random samples were taken outside the map boundaries, and is a contributing factor to why the algorithm required more random samples for this scenario compare to the previous scenarios.




(a) Tree of paths (green) between start and stop points for 1000 random samples (magenta)

(b) Shortest path (green) and the search space (black dashed ellipse)

Figure 8.10: Simulation result of an area with long straight corridors

### 8.2.5 Conclusion of the planning simulations

From the simulation results, it can be seen that the algorithm is capable of finding a path in various scenarios, and that for every path found, it is an improvement on the previous path found. It was noted that for most of the scenarios the planning algorithm was able to find a path within the first 50 samples of the map. The path found was also effective and did not include unnecessary turns. The scenario containing the long straight corridors proved to be a challenge to the planner and required many more samples before finding a path.

To further compare the performance of the algorithm, multiple iterations $i$ of a set number $n$ of random samples can be run and the following elements can be compared statistically or by a distribution function:

- The range of the length of paths found for each repetition $i$ of a specific map scenario.
- The total number of paths found for each repetition $i$ of a specific map scenario.
- Comparing after how many samples $n_i$ was the first path found in each $i$ iteration.
- The total time to sample $n$ random points across $i$ iterations.

These are just a few comparisons and ways to analyse the performance of the path planning algorithm and to present an evaluation thereof. Unfortunately these suggested evaluations could not be performed due to the time constraints of the project and the time spent on the implementation and practical aspects of the project.

# Chapter 9

# Conclusions

To conclude the thesis, the list of project objectives from Chapter 1 are provided and a summary follows of how each of them was addressed with references to the applicable chapters. The contribution of the work done in this project is mentioned and suggestions on work to follow are made.

## 9.1 Summary

The project is evaluated against the project objectives in Section 1.3, as listed below from:

- Model the dynamics, kinematics and behaviour of the vehicle to incorporate it in the path planner
- Improve the effectiveness and reliability of the control systems of the vehicle
- Set up static and complete environmental maps of actual terrain to execute a planned path and to test the navigation system
- Simplify the conflict detection algorithm such that it can process obstacles defined in the static environmental maps
- Develop a motion planning algorithm to plan a safe and good quality path
- Combine and implement the above mentioned into the AutoNav framework to test the path planning and execution thereof by the test vehicle.

The first and second objectives of the project were addressed in Chapter 3 with the modelling and control of the test vehicle and Chapter 4 that encapsulates the vehicle behaviour as manoeuvres to be used in the path planner. Each component in the vehicle drive train (except for the continuously variable transmission) was modelled and the vehicle speed controller was realised. The vehicle speed controller took the form of a cascade controller, with an inner-control loop to control the engine speed and an outer-control loop to control the wheel speed. Initially, the motivation for engine speed control was to enable control of the gear ratio of the variable transmission, but due to the inability to model

the transmission dynamically, the engine speed controller became trivial. The steering mechanism and disturbance loads effecting the steering actuator were modelled and an improvement on the previously implemented steering controller was achieved. Included in the steering control was the addition of a feed-forward input containing the steering rate requirement from the defined manoeuvres. This resulted in excellent tracking performance of the reference steering angle, at the possible cost of the steering actuator's lifetime. From the recommendations by the preceding project, it was already suggested to replace the low quality steering actuator with a higher quality actuator. The kinematic model of the vehicle was defined and produced the longitudinal and lateral dynamic equations used to calculate executable manoeuvres for the vehicle.

For the third objective, the theory and methods of presenting environmental maps to a motion planning algorithm is discussed in the first half of Chapter 5. The implementation and set-up of static and complete environmental maps followed in Chapter 7. Three maps representing three different testing sites were created on which to test the navigation system. The three testing sites were chosen for the unique type of terrain each of them offered: tarmac, gravel and grass. The boundary of the map was defined by the coordinates of each corner with straight lines connecting each corner. Non-drivable areas and obstacles within the map were represented by circles.

The simplification of the map representation was done to avoid unnecessary computational effort by the conflict detection module, which is listed as the fourth objective, and presented in the second half of Chapter 5. By representing obstacles in the map as circles, the vehicle pose (at what angle the vehicle approaches or passes by the obstacle) would not affect the possible conflict region. Furthermore, by expanding the obstacle area with the area of the vehicle[1] and reducing the vehicle area to a unit point, the conflict detection problem became a test whether a point lies within a circle.

The fifth objective, and also the main focus of the project, was approached by doing an extensive literature review in Chapter 6 of existing motion planning algorithms. The chosen method of searching the map is an adaptation of the RRT* algorithm that takes random samples of the defined search space. These random samples are then connected to a tree of reachable points that is built up from the starting point and expands towards the destination point. The path planning algorithm in this project improves on the existing planning methods by being able to guarantee that for every path found, it is an improvement on the previous path. It also provides the guarantee that the longer the planner is allowed to search, the more optimal the path becomes that it produces.

[1] with the use of Minkowski addition

These guarantees are a result of the following:

- Connections to a newly sampled random point are first sorted from the lowest to the highest cost, before testing the connections for conflict. The first conflict free connection that is returned, will then be the cheapest too.
- The search area from which randomly sampled points are taken is restricted by an ellipse that is a function of the path cost and which
- becomes smaller for every path found.

The last objective is accomplished by combining all of the different elements of the project together on the test vehicle, detailed in Chapter 7, and demonstrating a functional and successful path planning and autonomous execution of the planned path. The practical results achieved are presented in Chapter 8, which also includes simulation results to illustrate the functionality and capability of the path planning algorithm on scenarios like:

- Finding a path in a large open area with a small gateway.
- Finding a path in an area populated by multiple evenly spaced obstacles.
- Finding a path in long straight corridors.
- Comparing the ability to find a path between a given size map and the same map made ten times larger.

To finalise, the path planning algorithm that was developed, produced a safe and drivable path for the vehicle. The combined system and implementation was demonstrated successfully.

## 9.2 Contributions

The project contributed to improving the test vehicle controllers and provided insight on the dynamic modelling of a quad bike and its propulsion system. Without going in too much detail on the front suspension of the quad bike, the disturbances acting on the steering mechanism were defined and modelled. The kinematics of the quad bike were successfully encapsulated in the manoeuvres to represent a smooth trajectory of the vehicle, which is used in the path planning algorithm to plan an executable path for the vehicle.

Another contribution of the project was the development of a path planning algorithm that is an improvement on the classic RRT* algorithm. The improved path planner has the advantage to guarantee that it will find a more optimal path the longer it is allowed to search a given area.

## 9.3 Recommendations for future work

In work to follow on this project, it is recommended to remove the engine speed controller from the vehicle speed control loop. After realising that the dynamic modelling of the continuously variable transmission is not necessary to achieve sufficient control of the vehicle speed, the engine speed controller became unnecessary.

From the field test results it was noted that the cross-track error controller added high frequency control effort to the steering controller, and it is recommended to investigate the cause and improve on the cross-track error controller. The danger of this unnecessary control effort is that the steering actuator might draw excessive current for prolonged periods, and even with the current monitor implemented, the actuator might be damaged.

It was visually noted during the field tests, and can also be seen in the results, that whenever the quad bike makes a turn, the initial side slip on the front wheels is a lot more prominent than what was assumed. Even on the tarmac road, as the vehicle enters a turning manoeuvre and the steering angle changes in the direction of the turn, the vehicle tends to continue in a straight line for a short instance. This effect might be modelled as a time delay between the change in steer angle and the change in the heading of the vehicle. However, this effect is only prominent when the vehicle enters a turn, and not when exiting a turn.

Future work can include the expansion of the manoeuvre library. For this project the manoeuvres were limited to a constant vehicle speed, a constant steering angle rate of change and a set steer angle for all turns.

From the simulation results, it was evident that the path planning algorithm is capable of finding a path in various scenarios and can improve on the path found. It should be noted that the performance of the implemented path planner was not thoroughly analysed or measured against another path planning algorithm. Further work can include a detailed analysis on the performance of the algorithm.

In this project, the motion planning, conflict detection and mapping modules where implemented and executed in the MATLAB environment. Future work would be to port these modules to a more efficient programming language and to optimise the functions to pass pointers to data between them instead of copies of the data. For a large number of random points in a complex environment the tree of paths would become very large, requiring proper management of the memory resources on the executing platform. To conclude, the improved path planning algorithm was sufficient to demonstrate the implemented concepts, but it was not done with optimisation or time efficiency as a priority.

# Bibliography

[1] Aurenhammer, F. (1991). Voronoi diagrams - a survey of a fundamental geometric structure. *ACM Computing Surveys*, vol. 23, pp. 345–405.

[2] Baass, K. (1984). The use of clothoid templates in highway design. In: *Transportation Forum 1*.

[3] Barraquand, J. and Latombe, J.-C. (1990). A monte-carlo algorithm for path planning with many degrees of freedom. In: *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 1712–1717.

[4] Barraquand, J. and Latombe, J.-C. (1991). Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649.

[5] Beyers, C.J. (2012). *Motion planning algorithms for autonomous navigation for a rotary-wing UAV*. Master's Dissertation, Stellenbosch.

[6] Budynas, R.G. and Nisbett, J.K. (2008). *Shigley's Mechanical Engineering Design*. McGraw-Hill.

[7] Choset, H. and Burdick, J. (2000). Sensor-based exploration: the hierarchical generalized voronoi graph. *International Journal of Robotics Research*, vol. 19, no. 2, pp. 96–125.

[8] Choset, H., Walker, S., Eiamsa-Ard, K. and Burdick, J. (2000). Sensor-based exploration: incremental construction of the hierarchical generalized voronoi graph. *International Journal of Robotics Research*, vol. 19, no. 2, pp. 126–148.

[9] Company, K.Y.M. (2005). *Kymco Service Manual*. KWANG YANG Motor Co., Ltd.

[10] Connolly, C., Burns, J. and Weiss, R. (1990). Path planning using laplace's equation. In: *Proceedings of IEEE International Conference on Robotics and Automation*.

[11] Connor, D. and Krivodonova, L. (2010). *Interpolation of Two-Dimensional Curves with Euler Spirals*. Ph.D. thesis, University of Waterloo, Ontario, Canada.

[12] Delling, D., Sanders, P., Schultes, D. and Wagner, D. (2009). *Engineering route planning algorithms*, chap. Algorithmics of large and complex networks, pp. 117–139. Springer.

[13] Dragos, C., Preitl, S., Precup, R., Pirlea, D., Nes, C., Petriu, E.M. and Pozna, C. (2010). Modeling of a Vehicle with Continuously Variable Transmission. In: *2010 IEEE 19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD)*, pp. 441–446. ISBN 9781424468867.

[14] Ferguson, D., Likhachev, M. and Stentz, A. (2005). A guide to heuristic-based path planning. In: *Proceedings of ICAPS Workshop on Planning under Uncertainty for Autonomous Systems*.

[15] Frazzoli, E. (2001). *Robust Hybrid Control for Autonomous Vehicle Motion Planning*. Ph.D. thesis, Massachusetts Institute of Technology.

[16] Frazzoli, E., Dahleh, M. and Feron, E. (2000). Robust Hybrid Control for Autonomous Vehicle Motion Planning. In: *Proceedings of the IEEE Conference on Decision and Control*, pp. 821–826.

[17] Frazzoli, E., Dahleh, M.A. and Feron, E. (2005). Maneuver-Based Motion Planning for Nonlinear Systems With Symmetries. *IEEE Transactions on Robotics*, vol. 21, no. 6, pp. 1077–1091.

[18] Ge, S.S. and Cui, Y.J. (2002). Dynamic Motion Planning for Mobile Robots Using Potential Field Method. *Autonomous Robots*, vol. 13, no. 3, pp. 207–222.

[19] Genta, G. (1997). *Motor Vehicle Dynamics: Modeling and Simulation*. World Scientific.

[20] Geraerts, R. and Overmars, M.H. (2004). A Comparative Study of Probabilistic Roadmap Planners. In: Boissonnat, P.J.-D., Burdick, P.J., Goldberg, P.K. and Hutchinson, P.S. (eds.), *Algorithmic Foundations of Robotics V*, pp. 43–58. Springer Berlin Heidelberg. ISBN 978-3-540-45058-0.

[21] Goerzen, C., Kong, Z. and Mettler, B. (2009 November). A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance. *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, pp. 65–100. ISSN 0921-0296.

[22] Guillemin, V. and Pollack, A. (1974). *Differential Topology*. Prentice-Hall, New York.

[23] Hart, P.E., Nilsson, N.J. and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107.

[24] Howlett, J., Schulein, G. and Mansur, M. (2004). A practical approach to obstacle field route planning for unmanned rotorcraft. In: *American Helicopter Society 60th Annual Forum Proceedings, Baltimore, MD*.

[25] Hung, Y.H. and Hong, C.W. (2004). Bond graph dynamics of a rubber-belt continuously variable transmission. *International Journal of Vehicle Design*, vol. 35, no. 4, pp. 383–398.

[26] Jones, T. (2003). *Real-Time Probabilistic Collision Avoidance for Autonomous Vehicles, Using Order Reductive Conflict Metrics.* Ph.D. thesis, Massachusetts Institute of Technology.

[27] Jones, T. (2006). Tractable conflict risk accumulation in quadratic space for autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 1, pp. 39–48.

[28] Joubert, D. (2012). *Adaptive occupancy grid mapping with measurement and pose uncertainty.* Master's dissertation, Stellenbosch.

[29] Julió, G. and Plante, J.S. (2011 August). An experimentally-validated model of rubber-belt CVT mechanics. *Mechanism and Machine Theory*, vol. 46, no. 8, pp. 1037–1053. ISSN 0094114X.

[30] Kant, K. and Zucker, S. (1986). Toward efficient trajectory planning: The path-velocity decomposition. *International Journal of Robotics Research*, vol. 5, no. 3, pp. 72–89.

[31] Karaman, S. and Frazzoli, E. (2010). Incremental sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research.*

[32] Kavraki, L.E., Svestka, P., Latombe, J.-C. and Overmars, M.H. (1996). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580.

[33] Koenig, S. and Likhachev, M. (2002). Improved fast replanning for robot navigation in unknown terrain. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

[34] Lang, K. (2000). *Continuously Variable Transmissions: An overview of CVT research past, present and future.*

[35] Latombe, J.-C. (1991). *Robot Motion Planning.* Kluwer, Boston, MA.

[36] Lavalle, S.M. (1998). Rapidly-Exploring Random Trees: A New Tool for Path Planning. *TECHREPORT*, vol. 11.

[37] Lavalle, S.M. (2003). From Dynamic Programming to RRTs : Algorithmic Design of Feasible Trajectories. In: *Control Problems in Robotics*, pp. 19–37. Springer Berlin Heidelberg. ISBN 978-3-540-36224-1.

[38] Lavalle, S.M. (2006). *Planning Algorithms*. Cambridge University Press. ISBN 978-0521862059.

[39] LaValle, S.M. and Kuffner, J.J. (1999). Randomized Kinodynamic Planning. In: *1999 IEEE International Converence on Robotics and Automation*, pp. 473–479.

[40] Leonard, J. and Durrant-Whyte, H. (1991). Simultaneous map building and localization for an autonomous mobile robot. In: *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, pp. 1442–1447.

[41] Moravec, H. and Elfes, A. (1985). High resolution maps from wide angle sonar. In: *IEEE International Conference on Robotics and Automation*, pp. 116–121.

[42] Paielli, R. and Erzberger, H. (1997). Conflict probability estimation for free flight. *Journal of Guidance, Control, and Dynamics*, vol. 20, no. 3, pp. 588–596.

[43] Paielli, R. and Erzberger, H. (1999). Conflict probability estimation generalized to non-level flight. *Air Traffic Control Quarterly*, vol. 7, no. 3.

[44] Parmley, R. (2000). *Illustrated Sourcebook of Mechanical Components*. McGraw-Hill.

[45] Payeur, P., HÃ©bert, P., Laurendeau, D. and Gosselin, C. (1997). Probabilistic octree modeling of a 3d dynamic environment. In: *IEEE International Conference on Robotics and Automation*, pp. 1289–1296.

[46] Paynter, H. (1961). *Analysis and design of engineering systems*. MIT Press.

[47] Ramalingam, G. and Reps, T. (1996). An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, vol. 21, pp. 267–305.

[48] Rimon, E. and Koditschek, D. (1988). Exact robot navigation using cost functions: the case of distinct spherical boundaries in $E^n$. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

[49] Smith, R. and Cheeseman, P. (1986). Estimating uncertain spatial relationships in robotics. In: *Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence*, pp. 435–461.

[50] Smith, R. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68.

[51] Stentz, A. (1994). Optimal and Efficient Path Planning for Partially-Known Environments. In: *1994 IEEE International Conference on Robotics and Automation*, pp. 3310–3317.

[52] Stentz, A. (1995). The Focussed D * Algorithm for Real-Time Replanning. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, August, pp. 1652–1659. Morgan Kaufmann Publishers Inc.

[53] Thrun, S. (2002). *Exploring Artificial Intelligence In The New Millennium*, vol. Robotic mapping: a survey. Morgan Kaufmann.

[54] Tseng, C.-Y., Chen, L.-W., Lin, Y.-T. and Li, J.-Y. (2008). A hybrid dynamic simulation model for urban scooters with a mechanical-type CVT. In: *2008 IEEE International Conference on Automation and Logistics (ICAL)*, pp. 515–519. ISBN 9781424425037.

[55] Uicker, J., Pennock, G. and Shigley, J. (2003). *Theory of Machines and Mechanisms*. Oxford University Press.

[56] Visser, W. (2012). *Automation and Navigation of a Terrestrial Vehicle*. Master's Dissertation, Stellenbosch.

[57] Yang, L., Yang, J., Kuchar, J. and E., F. (2004). A real-time monte carlo implementation for computing probability of conflict. In: *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, pp. 617–631.